МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО
«___»_____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

Система аналізу настроїв тексту на основі тематичного моделювання

Спеціальність 121 Інженерія програмного забезпечення

Освітня програма «Інженерія програмного забезпечення»

Здобувач _____ Гліб ЧЕРНИГІН
«___»_____2024 р.

Керівник канд. техн. наук, доцент _____ Гліб ГОРБАНЬ
«___»_____2024 р.

Миколаїв 2024

Чорноморський національний університет імені Петра Могили

(повне найменування закладу вищої освіти)

Факультет	Комп'ютерних наук
Кафедра	Інженерії програмного забезпечення
Рівень вищої освіти	Другий (магістерський)
Освітній ступінь	Магістр
Спеціальність	121 Інженерія програмного забезпечення
Освітня програма	Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри інженерії
програмного забезпечення
_____Євген ДАВИДЕНКО
«____»_____2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу здобувача

_____Чернигіна Гліба Леонідовича_____

(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи
___Система аналізу настроїв тексту на основі тематичного моделювання___

Затверджена наказом ЧНУ ім. Петра Могили від «4» вересня 2024 р. №220

2. Строк представлення кваліфікаційної роботи «___»_____2024 р.

3. Очікуваним_результатом_є_система_аналізу_настроїв_тексту_на_основі тематичного_моделювання._____

4. Перелік питань, що підлягають розробці:
   - дослідження предметної області та аналіз існуючих аналогів ПЗ;
   - формування специфікації вимог до ПЗ;
   - визначення архітектури ПЗ;
   - моделювання та проєктування ПЗ;
   - розробка ПЗ;
   - здійснення тестування роботи ПЗ;
   - проведення аналізу результатів розробки;

5. Перелік графічних матеріалів:
   Презентація_____

Керівник роботи _____ __Гліб ГОРБАНЬ___
Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

Здобувач _____ _Гліб ЧЕРНИГІН__
Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

Дата видачі завдання «5» вересня 2024р

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема:_____Система аналізу настроїв тексту на основі тематичного моделювання_____
_____

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КМР	02.09.2024 р.	02.09.2024 р.	виконано
2.	Огляд літератури за темою роботи	03.09.2024 р.	06.09.2024 р.	виконано
3.	Складання календарного плану КМР	09.09.2024 р	09.09.2024 р	виконано
4.	Аналіз предметної області	10.09.2024 р.	13.09.2024 р.	виконано
5.	Розробка проєктних рішень	16.09.2024 р.	18.09.2024 р.	виконано
6.	Моделювання та конструювання ПЗ	19.09.2024 р.	04.10.2024 р.	виконано
7.	Кодування, тестування та апробація розробленого ПЗ, аналіз результатів тестування, розробка керівництва користувача	07.10.2024 р.	18.11.2024 р.	виконано
8.	Відгук керівника КМР	18.11.2023 р.	18.11.2024 р.	виконано
9.	Оформлення КМР та презентації	19.11.2024 р.	22.11.2024 р.	виконано
10.	Попередній захист	28.11.2024 р.	28.11.2024 р.	виконано
11.	Рецензування	29.11.2024 р.	3.12.2024 р.	виконано
12.	Завершення оформлення КМР та презентації	3.12.2024 р.	12.12.2024 р.	виконано
13.	Захист кваліфікаційної роботи	19.12.2024 р.	20.12.2024 р.	виконано

Розробив здобувач _____Чернигін Г. Л._____ _____

(прізвище, ім'я, по батькові) *(підпис)*

«__» _____ 2024 р.

Керівник роботи __канд. техн. наук, доцент Горбань_ Г. В.__ _____

(*посада, прізвище, ім'я, по батькові)* *(підпис)*

«__» _____ 2024 р.

АНОТАЦІЯ

до кваліфікаційної магістерської роботи

«Система аналізу настроїв тексту на основі тематичного моделювання»

Здобувач 608 гр.: Чернигін Г. Л.

Керівник: зав. кафедри ІПЗ, канд. техн. наук, доцент Горбань Г. В.

Дана робота присвячена розробці системи аналізу настроїв тексту на основі методів машинного навчання SVC та LDA для автоматизації процесу аналізу настроїв текстів.

Об’єкт: процес аналізу текстових даних для виявлення настроїв та тем.

Предмет: програмні засоби створення системи для аналізу настроїв тексту на основі тематичного моделювання.

Метою роботи є підвищення ефективності аналізу настроїв у тексті та тематичного моделювання шляхом розробки системи системи аналізу настроїв на основі тематичного моделювання.

Для досягнення визначеної мети необхідно вирішити наступні **завдання**:

- аналіз існуючих аналогів ПЗ;
- формування специфікації вимог до програмного забезпечення;
- створення блок-схем та діаграм роботи застосунку;
- побудова архітектури системи;
- збір та аналіз методичних рекомендацій;
- розробка машинної моделі з використанням Python;
- розробка frontend-частини вебзастосунку на базі фреймворку Angular;
- розробка backend-частини вебзастосунку на базі фреймворку ASP.NET.

Пояснювальна записка кваліфікаційної магістерської роботи складається з вступу, чотирьох розділів, висновків та додатків.

У вступі обґрунтовано актуальність теми, сформульовано мету і завдання роботи, а також визначено наукову новизну та практичну значущість.

У першому розділі здійснено огляд предметної області, досліджено сучасні методи аналізу тексту, а також проаналізовано наявні програмні рішення, виявивши їхні переваги та недоліки та створено специфікацію системи.

У другому розділі змодельовані сценарції використання, створені алгоритми роботи програмного забезпечення та діаграму розгортання.

У третьому розділі створені UML-діаграми класів та проєктів, спроєктований інтерфейс системи, а також розглянутий стек технологій.

У четвертому розділі представлено програмну реалізацію системи, яка складається з бекенд-частини на основі ASP.NET Core, фронтенд-інтерфейсу на Angular, а також інтеграції авторизації через Keycloak.

У висновках проводиться аналіз виконаного об’єму робіт та отриманих у ході виконання результатів.

Кваліфікаційна робота містить **89** сторінок основної частини, **4** розділи, **39** рисунків, **8** таблиць, **30** джерел в переліку посилань та **2** додатки.

Ключові слова: *аналіз тексту, машинне навчання, аналіз настроїв, тематичне моделювання, SVC, LDA, ASP.NET Core, Angular, Keycloak, FastAPI.*

ABSTRACT

to the qualification master's thesis

"Text mood analysis system based on thematic modeling"

Student of group 608: Chernyhin H. C.

Supervisor: Head of the Department of Software Engineering, PhD, Associate Professor Horban H. V.

This work is devoted to developing a text mood analysis system based on machine learning methods SVC and LDA to automate the process of text mood analysis.

Object: the process of text data analysis to identify moods and topics.

Subject: software tools for creating a system for text mood analysis based on thematic modeling.

The work aims to increase the efficiency of text mood analysis and thematic modeling by developing a system of mood analysis based on thematic modeling.
To achieve this goal, the following tasks need to be solved:

- analysis of existing software analogs;
- forming a specification of software requirements;
- creation of flowcharts and diagrams of the application;
- building the system architecture;
- collection and analysis of methodological recommendations;
- development of a machine model using Python;
- development of the frontend part of the web application based on the Angular framework;
- development of the backend part of the web application based on the ASP.NET framework.

The explanatory note of the qualification master's thesis consists of an introduction, four sections, conclusions, and appendices.

The introduction substantiates the topic's relevance, formulates the work's goal and objectives, and determines the scientific novelty and practical significance.

The first section reviews the subject area, investigates modern text analysis methods, analyzes existing software solutions, identifies their advantages and disadvantages, and creates a system specification.

The second section models usage scenarios and creates software algorithms and a deployment diagram.

The third section creates UML diagrams of classes and projects, designs the system interface, and examines the technology stack.

The fourth section presents the system's software implementation, which consists of a backend based on ASP.NET Core, a frontend interface based on Angular, and authorization integration via Keycloak.

The conclusions analyze the scope of work performed and the results obtained during the implementation.

The qualification work contains **89** pages of the main part, **4** sections, **39** figures, **8** tables, **30** sources in the list of references, and **2** appendices.

Keywords: *text analysis, machine learning, sentiment analysis, topic modeling, SVC, LDA, ASP.NET Core, Angular, Keycloak, FastAPI.*

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД	–	база даних
ОС	–	операційна система
ПЗ	–	програмне забезпечення
ПК	–	персональний комп’ютер
API	–	application programming interface
CSV	–	Comma-separated values
JS	–	JavaScript
LDA	–	Latent Dirichlet allocation
ML	–	Machine Learning
ORM	–	Object-relational mapping
REST	–	Representational state transfer
SVC	–	Support Vector Classifier
UI	–	user interface
UML	–	unified modeling language
UX	–	user experience

ВСТУП

У сучасному світі інформаційні потоки збільшуються з шаленою швидкістю, що вимагає нових підходів до аналізу даних. Одним із перспективних напрямків є аналіз настроїв тексту, що дозволяє отримати важливі уявлення про думки, емоції та реакції користувачів. Такий аналіз є корисним для різноманітних галузей, включаючи маркетинг, соціологію, політологію та інші сфери, де важливо розуміти суспільні настрої та реакції на певні події.

Завдяки сучасним технологіям машинного навчання та обробки природної мови стало можливим автоматизувати процес аналізу текстів на основі їх змісту та тональності. Це дозволяє не тільки значно прискорити процес обробки інформації, але й покращити точність отриманих результатів, що є важливим у контексті аналізу великих обсягів даних. Інструменти тематичного моделювання можуть виявляти приховані теми та ключові елементи в текстах, що створює нові можливості для аналізу настроїв у динамічному інформаційному просторі.

Актуальність: Актуальність теми магістерської кваліфікаційної роботи зумовлена зростаючою необхідністю в автоматизованому аналізі великих обсягів текстових даних. Аналіз настроїв у поєднанні з тематичним моделюванням дає можливість краще розуміти тенденції в суспільстві, що є цінним для багатьох зацікавлених сторін, таких як бізнес, політика та наука.

Мета: підвищення ефективності обробки текстових даних шляхом розробки системи, яка використовує методи аналізу настроїв та тематичного моделювання для виявлення ключових елементів змісту.

Об’єкт: Об’єктом дослідження є процес аналізу текстових даних для виявлення настроїв та тем.

Предмет: Предметом дослідження є програмні засоби створення системи для аналізу настроїв тексту на основі тематичного моделювання.

Сфера застосування: Результуючий програмний продукт можна використовувати для аналізу текстів у різних сферах, таких як маркетинг, соціальні дослідження, політичний аналіз тощо.

Для досягнення визначеної мети необхідно вирішити наступні **завдання**:

– аналіз існуючих аналогів ПЗ;
– формування специфікації вимог до програмного забезпечення;
– створення блок-схем та діаграм роботи застосунку;
– побудова архітектури системи;
– збір та аналіз методичних рекомендацій;
– розробка машинної моделі з використанням Python;
– розробка frontend-частини вебзастосунку на базі фреймворку Angular;
– розробка backend-частини вебзастосунку на базі фреймворку ASP.NET.

Апробація результатів КМР відбулась під час XXVII Всеукраїнської науково-практичної конференції «Могилянські читання – 2024» (Миколаїв, 06-10 листопада 2024 р.) (Додаток А).

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд методів аналізу настроїв та тематичного моделювання

Аналіз настроїв (sentiment analysis) і тематичного моделювання (topic modeling) – комплексні завдання, що стали актуальними з появою соціальних мереж і месенджерів.

Аналіз настроїв – це процес автоматичного виявлення та класифікації емоційного тону тексту. Головна мета полягає в тому, щоб визначити, чи позитивний, чи негативний, чи нейтральний емоційний відтінок міститься у тексті. Цей аналіз може бути застосований до різних типів текстуальних даних, таких як соціальні медіа, відгуки клієнтів, новини, відгуки товарів тощо. У випадку тематики роботи, даний метод застосовуватиметься до дописів та коментарів у каналі.

Класичні методи машинного навчання відіграють важливу роль у задачах аналізу настроїв, пропонуючи різноманітні підходи до обробки тексту. Вони варіюються від простих і легких у реалізації моделей до більш складних алгоритмів, здатних розв’язувати нелінійні задачі. Кожен метод має свої сильні сторони та обмеження, що робить їх застосування контекстуально залежним від особливостей даних та вимог до точності класифікації. Розглянемо найпопулярніші методи для аналізу настроїв:

Наївний баєсівський класифікатор (Naive Bayes Classifier) є одним із найпростіших і водночас найефективніших алгоритмів для аналізу настроїв, заснованим на теоремі Байєса [1]. Його основна перевага — швидкість і ефективність у роботі з великими текстовими наборами даних, оскільки він працює на основі припущення, що всі ознаки (слова в тексті) незалежні одна від одної. Це робить його менш вимогливим до обчислювальних ресурсів і простим у реалізації. Однак наївний баєсівський підхід має істотний недолік: припущення

незалежності ознак часто є нереалістичним у природній мові. Це призводить до того, що алгоритм не завжди добре враховує контекст слів, особливо у випадках складних або довгих текстів, що може впливати на точність класифікації.

Метод опорних векторів (Support Vector Machines) є більш складним і потужним методом для аналізу настроїв, особливо коли дані є добре структурованими [2]. Однією з головних переваг SVM є здатність розв’язувати лінійно та нелінійно розділювані задачі, що дозволяє ефективно класифікувати текстові дані навіть у складних випадках. Завдяки використанню ядрових функцій SVM може працювати з великими обсягами текстових даних і добре справлятися з високовимірними векторними просторами, які виникають під час текстового аналізу. Недоліком SVM є його потреба у великих обчислювальних ресурсах та складність налаштування для досягнення оптимальних результатів, особливо для нелінійних задач. Крім того, для інтерпретації результатів SVM потрібно більше зусиль, оскільки цей метод працює на основі максимальної межі поділу, що ускладнює розуміння того, чому модель прийняла конкретне рішення.

Дерева рішень (Decision Trees) є одним із найбільш інтуїтивно зрозумілих методів для аналізу настроїв [3]. Їх основна перевага полягає в тому, що вони легко інтерпретуються: кожне рішення в дереві можна просто пояснити за допомогою логічних правил, що дозволяє користувачам легко зрозуміти, як модель приймає рішення. Крім того, дерева рішень добре працюють на невеликих наборах даних і здатні обробляти як числові, так і категоріальні дані. Проте основним недоліком дерев рішень є їх схильність до переобучення, особливо коли дерева стають занадто глибокими. Це призводить до низької узагальнюючої здатності моделі, що може погіршити її продуктивність на нових даних. Також дерева можуть бути не дуже ефективними при роботі з великими текстовими корпусами, особливо якщо структура даних складна.

Логістична регресія (Logistic Regression) є простим, але ефективним методом для бінарної класифікації, зокрема й для аналізу настроїв тексту [4]. Вона добре підходить для задач, де потрібно передбачити дві категорії (наприклад, позитивний чи негативний настрій). Основною перевагою логістичної регресії є її швидкість, простота та здатність добре працювати на невеликих наборах даних. Вона також надає можливість вимірювати ймовірність належності тексту до кожної з категорій, що є корисним у багатьох практичних застосуваннях. Проте, її основний недолік полягає в тому, що цей метод погано працює зі складними нелінійними даними. Якщо настрій тексту сильно залежить від контексту або взаємодії між словами, логістична регресія може бути недостатньо точною.

Тематичне моделювання – це метод аналізу тексту, який дозволяє виявити та ідентифікувати теми, що присутні у наборі текстових даних [5]. Головна ідея полягає в тому, щоб кожен документ міг бути представлений як комбінація різних тем з певними ймовірностями. Даний метод аналізу тексту, у комбінації з аналізом настроїв, дозволить проаналізувати та виявити динаміку спільноти.

Основні методи тематичного моделювання включають Latent Dirichlet Allocation (LDA), Non-Negative Matrix Factorization (NMF), та сучасні нейронні підходи, такі як BERT Topic Modeling. Кожен із цих методів має свої особливості й застосування.

Latent Dirichlet Allocation (LDA) є генеративною моделлю, яка припускає, що кожен документ є сумішшю тем, а кожна тема є сумішшю слів[6]. Основна ідея LDA полягає в тому, щоб віднайти ці теми, які найкраще описують структуру текстових даних. Модель використовує ієрархічний підхід для визначення тем на основі частоти слів у документах, що дозволяє їй автоматично виявляти ймовірнісні розподіли тем у кожному документі. Однією з основних переваг LDA є його здатність обробляти великі обсяги текстових даних і виділяти теми без необхідності попереднього визначення кількості тем. Проте, LDA також має свої

обмеження, зокрема, потребу в налаштуванні гіперпараметрів, які можуть суттєво впливати на результати, а також труднощі в інтерпретації тем, якщо їх кількість не відповідає реальній структурі даних.

Іншим популярним методом є **Non-Negative Matrix Factorization** (NMF), який також використовується для тематичного моделювання [7]. NMF розглядає текстові дані як матрицю терміни-документи і розкладає цю матрицю на дві не від'ємні матриці – одну для терміни-теми і іншу для теми-документи. Таким чином, NMF допомагає виявити латентні теми, які пояснюють спостережувані терміни у документах. Однією з основних переваг NMF є його здатність до виявлення інтерпретованих тем завдяки обмеженню на не від'ємність, що забезпечує більш зрозумілі та узгоджені результати. Проте NMF може бути чутливим до початкових умов і потребує попереднього визначення кількості тем, що може ускладнити налаштування моделі.

Latent Semantic Analysis (LSA), або латентний семантичний аналіз, є ще одним методом, який базується на матричному розкладанні [8]. LSA використовує сингулярне розкладання матриці терміни-документи для зменшення розмірності і виділення латентних семантичних структур у текстових даних. Це дозволяє моделі виявляти синонімічні та семантичні зв'язки між словами, що допомагає краще розуміти контекст тем. Перевага LSA полягає в його здатності зменшувати шум у даних і виявляти приховані теми. Однак, LSA може іноді генерувати теми, які важко інтерпретувати, а також має обмеження у відображенні контекстуальних зв'язків на базі статичних матриць.

Сучасні моделі, такі як **BERT** (Bidirectional Encoder Representations from Transformers), пропонують нові підходи до тематичного моделювання. BERT використовує трансформери для глибокого контекстуального розуміння тексту і може бути адаптований для тематичного аналізу за допомогою специфічних налаштувань або тонкої налаштування на задачі тематичного моделювання [9].

Перевага BERT полягає в його здатності враховувати контекст слів з обох сторін і забезпечувати точні результати для складних завдань, але цей метод потребує значних обчислювальних ресурсів і великого обсягу даних для навчання.

Таким чином, методи аналізу настроїв та тематичного моделювання в машинному навчанні надають різноманітні способи для виявлення та організації тем у текстових даних. Вибір конкретного методу залежить від специфіки завдання, обсягу даних та бажаної точності в результатах.

1.2 Огляд систем-аналогів

Аналіз існуючих аналогів є важливою складовою під час розробки будь-якої нової системи, зокрема систем для аналізу настроїв тексту на основі тематичного моделювання. Це дослідження дозволяє зрозуміти, які рішення вже існують на ринку, які функції вони надають та наскільки ефективно ці рішення відповідають потребам користувачів. Оцінка конкурентів дає можливість визначити найкращі практики в реалізації таких систем, що дозволяє уникати зайвих помилок та неефективних рішень на етапі розробки.

Крім того, аналіз аналогів допомагає виявити ключові функціональні вимоги, необхідні для задоволення користувацьких очікувань. Досліджуючи можливості конкуруючих систем, можна зрозуміти, які функції є базовими для систем аналізу настроїв, наприклад, обробка великих обсягів тексту, підтримка різних мов, точність у визначенні емоційної тональності та можливість інтеграції з іншими платформами. Це також дає змогу визначити, які функції ще не реалізовані або недостатньо розвинені в існуючих рішеннях, і розробити стратегії для їх вдосконалення в новій системі.

Аналіз недоліків конкурентів є ще одним критично важливим етапом у цьому процесі. Вивчення слабких сторін наявних систем дозволяє не лише уникнути повторення помилок, але й забезпечити конкурентну перевагу новій

розробці. Наприклад, якщо існуючі системи мають проблеми зі швидкістю обробки даних або не забезпечують достатньо точний аналіз настроїв, це відкриває можливості для впровадження інноваційних рішень, які підвищать продуктивність та якість аналізу. Усунення цих недоліків дозволить створити більш конкурентоспроможний продукт, що краще задовольняє потреби користувачів.

У якості основних систем-аналогів для цього обрано аналоги: Medallia (табл. 1.1), Lexalytics (табл. 1.2), Clarabridge (табл. 1.3).

Medallia

Medallia − платформа, яка пропонує широкий спектр інструментів для аналізу тексту, включаючи аналіз настроїв, класифікацію тексту, витяг сутностей та інші завдання [10]. Система використовує методи машинного навчання для автоматизації аналізу тексту.

Таблиця 1.1 – Опис застосунку

Назва	**Medallia**
Архітектура	Хмарна архітектура з можливістю інтеграції через API
Розробник	MonkeyLearn Inc.
Мова реалізації	Python, JavaScript
Функції	1. Аналіз настроїв. 2. Класифікація тексту. 3. Тематичне моделювання. 4. Витяг сутностей.
Переваги	1. Висока точність 2. налаштовувані моделі для аналізу тексту 3. масштабованість для великих обсягів даних
Недоліки	1. Дорога ціна для великих обсягів даних. 2. Обмеженість безкоштовної версії. 3. Потреба в підготовці моделей для специфічних завдань.

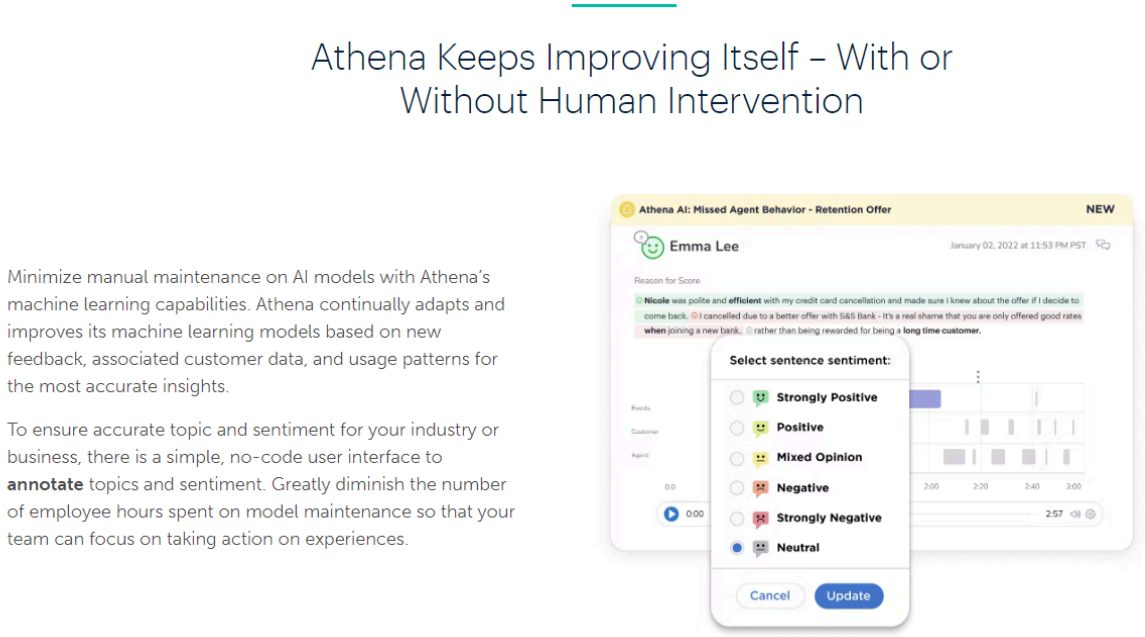

Рисунок 1.1 – Вигляд інтерфейсу Medallia

Lexalytics

Lexalytics – це ще одна потужна платформа для аналізу настроїв і тематичного аналізу [11]. Система використовує обробку природної мови (NLP) та машинне навчання для аналізу тексту, виявлення ключових тем і тональності повідомлень.

Lexalytics орієнтована на великий бізнес і надає гнучкі можливості для аналізу соціальних медіа, оглядів, документів та інших джерел текстової інформації. Система також інтегрує можливості аналізу тенденцій на основі тематики.

Таблиця 1.2 – Опис застосунку

Назва	**Lexalytics**
Архітектура	Локальні та хмарні рішення, інтеграція з різними базами даних і API
Розробник	Lexalytics, Inc.
Мова реалізації	Python, Java, C++
Функції	1. Аналіз настроїв. 2. Тематичний аналіз 3. Аналіз соціальних медіа.
Переваги	1. Можливість локального розгортання. 2. Підтримка кількох мов. 3. Аналіз соціальних медіа.
Недоліки	1. Дорога ціна для великих обсягів даних. 2. Обмеженість безкоштовної версії. 3. Потреба в підготовці моделей для специфічних завдань.

Рисунок 1.2 – Секція з лендінгу Lexalytics

Clarabridge

Clarabridge – це корпоративна платформа для аналізу тексту, яка дозволяє проводити багаторівневий аналіз настроїв, визначати тематику та ключові емоції в текстах [12]. Clarabridge також спеціалізується на аналізі відгуків клієнтів у різних галузях, таких як роздрібна торгівля, охорона здоров'я та фінанси. Вона підтримує аналіз текстів із соціальних медіа, опитувань та інших джерел, що робить її дуже корисною для комплексної оцінки настроїв у різних сферах.

Таблиця 1.3 – Опис застосунку

Назва	**Clarabridge**
Архітектура	Хмарна архітектура з модульною структурою
Розробник	Clarabridge, Inc.
Мова реалізації	Python, Java
Функції	1. Аналіз настроїв. 2. Виявлення тем та аналіз емоцій. Аналіз соціальних медіа. 3. Інтеграція з системами управління досвідом клієнтів.
Переваги	1. Багатомовна підтримка. 2. Інтеграція з клієнтськими базами даних.
Недоліки	1. Висока вартість для підприємств. 2. Обмежена підтримка для малого бізнесу. 3. Складний процес налаштування для нетехнічних користувачів.

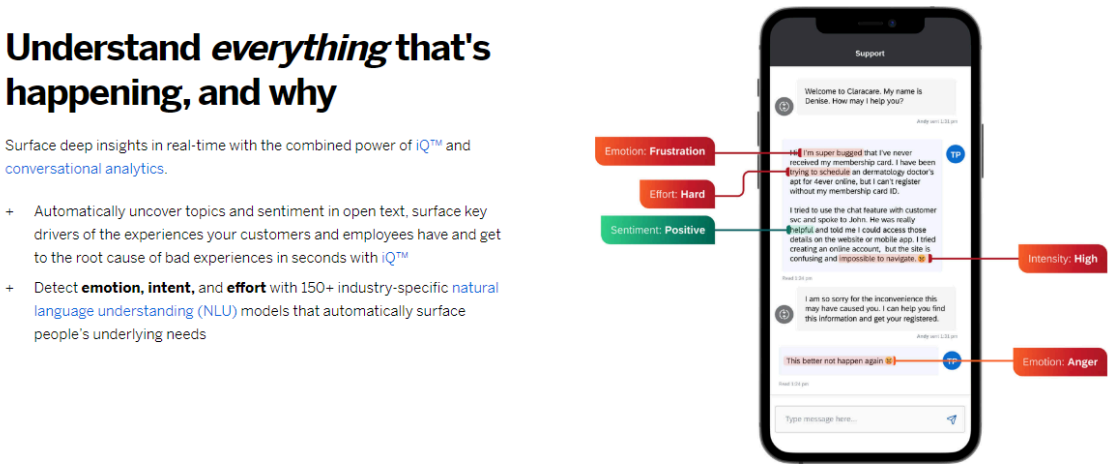

Рисунок 1.3 – Секція з лендінгу Clarabridge

Вказані аналоги надають функціонал з обробки та аналізу настроїв текстів, проте відрізняються сферою застосування та можливістю інтеграції та локального розгортання. Аналіз цих систем дозволив виділити основні пріоритети для розробки майбутньої системи, що базуються на недоліках аналогів.

1.3 Аналіз системи, що розробляється

Система, що розробляється, призначена для оптимізації процесу аналізу настроїв тексту та тематичного моделювання. Аналіз системи включає в себе огляд існуючих проблем в сфері аналізу настроїв і тематичного моделювання та визначення ключових вимог, які мають бути враховані при розробці.

Таблиця 1.4 – Опис системи, що розробляється

Функції	1. Реєстрація. 2. Авторизація. 3. Аналіз настроїв тексту та тематичне моделювання. 4. Інтеграція з іншими системами. 5. Візуалізація даних та експорт результатів.
Користувачі	1. Адміністратор. 2. Аналітики.
Сценарії роботи	1. Користувач завантажує текстові дані в систему, яка автоматично класифікує тексти за полярністю (позитивна, негативна, нейтральна). 2. Користувач завантажує текстовий корпус для виявлення основних тем, що повторюються, і отримує відображення тем в графічному або табличному вигляді. 3. Користувач вводить текст або завантажує документ, система автоматично виявляє та виводить ключові слова та фрази. 4. Користувач налаштовує інтеграцію з CRM або іншими системами для завантаження даних для аналізу. 5. Користувач отримує візуалізації (графіки, діаграми) для глибшого розуміння розподілу настроїв і тем у текстових даних. 6. Користувач може експортувати результати аналізу в PDF форматі для подальшого використання. 7. Адміністратор контролює процеси моніторингу, оновлення даних і забезпечення їхньої актуальності.

Розроблювана система для аналізу настроїв тексту на основі тематичного моделювання пропонує ключові функції, такі як класифікація настроїв, виявлення тем і візуалізація даних. Ці можливості дозволяють користувачам ефективно обробляти та аналізувати текстову інформацію, автоматично визначаючи полярність тексту та основні теми. Система також підтримує інтеграцію з популярними соціальними платформами і має функцію експорту результатів у різних форматах.

1.4 Специфікація вимог системи

ПРИЗНАЧЕННЯ ТА МЕЖІ ПРОЄКТУ

Призначення системи (застосунку), для якої розробляється програмне забезпечення

Система спрямована на автоматизацію процесу аналізу настроїв тексту на основі тематичного моделювання та генерації звітів щодо результатів аналізу.

Погодження, що ухвалені в програмній документації

Принципи, закладені в програмній документації, охоплюють опис ключових функцій системи, методів машинного навчання для обробки та представлення даних, а також вимоги до користувацького інтерфейсу. Затверджено алгоритми автоматизованого аналізу тексту та процеси створення звітів.

Межі проєкту ПЗ

Крайня дата завершення роботи над ПЗ – 12.12.2024 р.

ЗАГАЛЬНИЙ ОПИС

Сфера застосування

Результуючий програмний продукт можна використовувати для аналізу текстів у різних сферах, таких як маркетинг, соціальні дослідження, політичний аналіз тощо.

Характеристики користувачів

Користувачі системи включають в себе соціальних дослідників, аналітиків, а також маркетологів та рекламодавців. Усі групи користувачів системи мають спільну потребу в автоматизації та точності аналізу текстових даних. Для роботи з системою користувачам необхідно мати персональні комп’ютери або ноутбуки, оснащені веббраузером та підключені до Інтернету. Система має бути адаптована для різних типів пристроїв і має забезпечувати інтуїтивно зрозумілий і зручний користувацький інтерфейс, що гарантує комфортну взаємодію незалежно від обладнання.

Загальна структура і склад системи

Основні частини для створення ПЗ включають в себе: проксі-сервер, БД, вебзастосунок, API.

Загальні обмеження

Обмеження для роботи з ПЗ – наявність доступу до мережі Інтернет та наявність відповідної версії браузера.

ФУНКЦІЇ СИСТЕМИ АНАЛІЗУ НАСТРОЇВ ТЕКСТУ НА ОСНОВІ ТЕМАТИЧНОГО МОДЕЛЮВАННЯ

Функція аналізу тексту

Опис функції

Функція аналізу настроїв тексту дозволяє автоматично визначати емоційне забарвлення текстових даних та основні теми тексту.

Вхідна і вихідна інформація

Вхідна інформація – текстові дані;

вихідна інформація – результат аналізу тексту у вигляді звітів або візуалізацій.

Функціональні вимоги

Система повинна надавати можливість завантаження тексту та генерувати звіти з класифікацією настроїв.

Функція експорту результатів аналізу

Опис функції

Функція експорту результатів аналізу забезпечує можливість експорту даних, отриманих внаслідок аналізу настроїв і тематичного моделювання, у різні формати для подальшого використання або зберігання.

Вхідна і вихідна інформація

Вхідна інформація – результати аналізу тексту;

вихідна інформація – Експортовані дані у форматі PDF, що містять звіт про результати аналізу.

Фунціональні вимоги

Система повинна дозволяти експорт результатів в формат PDF, забезпечувати можливість експорту у фоновому режимі та повідомляти користувача про завершення експорту

ВИМОГИ ДО ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ.

Архітектура програмної системи

Архітектура складається з клієнтської частини, серверної частини та БД.

Системне програмне забезпечення

Розробка front-end частини повинна бути реалізована з використання фреймворку Angular, для back-end частини – ASP.NET та PostgreSQL у якості БД. Для тренування моделей машинного навчання повинен бути використаний Python з бібліотеками ntlk, sklearn, spacy та pandas. Натреновані моделі мають бути експортовані у форматі Open Neural Network Exchange (ONNX) та у подальному використані на back-end частині. Використання ШІ здійснено за допомогою стороннього сервісу OpenAI і має використовуватись для конкретних функціональних можливостей системи.

Мережне програмне забезпечення

Для створення ПЗ використано ОС Windows 11, у якості інтегрованого середовища розробки – vscode та Rider та Firefox у якості браузеру.

Програмне забезпечення ведення інформаційної бази

Усе введення інформаційної бази відбувається лише через REST-інтерфейс, який надає back-end частина, що з PostgreSQL за допомогою ORM-бібліотеки Entity Framework.

Мова і технологія розробки ПЗ

Для розробки backend частини програмного забезпечення має використовуватись C# та ASP.NET, для тренування моделей машинного навчання має використовуватись Python.

Для розробки frontend частини програмного забезпечення має використовуватись Typescript та Angular.

ВИМОГИ ДО ЗОВНІШНІХ ІНТЕРФЕЙСІВ

Інтерфейс користувача

Інтерфейс користувача системи повинен бути інтуїтивно зрозумілим і простим у використанні, забезпечувати зручний доступ до всіх основних функцій, таких як завантаження даних, налаштування параметрів аналізу та перегляд результатів. Інтерфейс має включати чіткі інструкції, добре організовані меню та інструменти для візуалізації даних, щоб користувач міг легко взаємодіяти з системою без необхідності додаткового навчання.

Апаратний інтерфейс

Система повинна бути сумісна з персональними комп’ютерами та ноутбуками, оснащеними сучасними веббраузерами та підключеними до Інтернету. Вона має бути оптимізована для роботи на різних типах пристроїв, включаючи різні операційні системи та роздільні здатності екранів, щоб забезпечити стабільну продуктивність і безперебійний доступ до всіх функцій незалежно від використовуваного обладнання.

Програмний інтерфейс

Програмний інтерфейс системи повинен підтримувати стандартні протоколи та API з чітко визначеними методами для доступу до функцій аналізу, експорт результатів та управління даними.

Комунікаційний протокол

Застосунок передбачає використання протоколів HTTP, HTTPS та TCP/IP.

ВЛАСТИВОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Доступність

Система повинна бути доступною для користувачів у режимі 24/7 з мінімальними перервами на технічне обслуговування. Доступ до системи має бути забезпечено через веббраузер з будь-якого сучасного пристрою, що підключений до Інтернету.

Супроводжуваність

Система повинна бути легкою для супроводження та оновлення. Вона повинна містити зрозумілу документацію для адміністраторів та розробників, що описує процеси оновлення, виправлення помилок і розширення функціоналу. Код повинен бути добре структурованим і коментованим, що спрощує процес усунення помилок і впровадження нових функцій.

Переносимість

Система повинна бути переносимою між різними операційними системами та платформами, такими як Windows, macOS та Linux. Вона має бути здатна до адаптації під різні версії веббраузерів та апаратних конфігурацій без значних змін у функціональності.

Продуктивність

Система повинна забезпечувати високу продуктивність, що включає швидкість обробки великих обсягів даних і зменшення часу затримки при виконанні запитів. Вона повинна ефективно використовувати системні ресурси

для забезпечення швидкого та безперебійного виконання всіх основних функцій без перевантаження системи.

Надійність

Система повинна бути надійною, з високим рівнем доступності та стабільності. Вона повинна забезпечувати коректну роботу навіть при високих навантаженнях і в разі збою одного з компонентів. Надійність також включає регулярне резервне копіювання даних і можливість швидкого відновлення системи в разі критичних помилок.

Безпека

Система повинна забезпечувати високий рівень безпеки для захисту даних і конфіденційної інформації. Це включає захист від несанкціонованого доступу, шифрування даних при передачі та зберіганні, а також регулярні перевірки на вразливості та застосування оновлень безпеки. Користувачі повинні мати можливість використовувати багатофакторну аутентифікацію для підвищення рівня захисту доступу.

Висновки до розділу 1

В першому розділі кваліфікаційної роботи магістра розглянуто популярні методи аналізу настроїв та тематичного моделювання з їх перевагами і недоліками, що дає глибше розуміння того, які методи варто обрати для розробляємої системи.

Проведено аналіз систем-аналогів для виявлення їх слабких та сильних сторін, що дозволяє внести покращення у мабутню систему та уникнути помилок конкурентів.

Розроблено специфікацію вимог до системи на основі проаналізованих даних, яка включає функціональні та нефункціональні вимоги до системи та архітектуру.

2 МОДЕЛЮВАННЯ СИСТЕМИ АНАЛІЗУ НАСТРОЇВ ТЕКСТУ НА ОСНОВІ ТЕМАТИЧНОГО МОДЕЛЮВАННЯ

2.1 Моделі системи

ERD (Entity Relationship Diagram) — це діаграма, яка графічно представляє сутності (entity) системи та їхні взаємозв'язки [13]. Вона використовується для моделювання бази даних і є основним інструментом при розробці інформаційних систем, що дозволяє візуалізувати структуру даних і взаємодію між окремими елементами. Кожна сутність у системі відображає окремий об'єкт або поняття, яке зберігає певні атрибути.

ERD (рис. 2.1) демонструє зв'язки між основними сутностями системи: «Користувач», «Запит на аналіз», «Аналіз настрою», «Тематичне моделювання» та «Звіт з аналізу».

Основні сутності системи аналізу настроїв тексту на основі тематичного моделювання включають в себе наступні атрибути:

– сутність «Користувач»: атрибути «User» – id (uuid, первинний ключ), username (varchar, 100), created_at (timestamp).

– сутність «Запит на аналіз»: атрибути «AnalysisRequest» – id (uuid, первинний ключ), user_id (uuid, зовнішній ключ на User.id), corpus (json), parent_request (uuid, зовнішній ключ на AnalysisRequest.id), requested_at (timestamp).

– сутність «Аналіз настрою»: атрибути «SentimentAnalysis» – analysis_id (uuid, первинний ключ), text_id (uuid, зовнішній ключ на AnalysisRequest.id), sentiment_result (json).

– сутність «Тематичне моделювання»: атрибути «TopicModeling» – id (uuid, первинний ключ), text_id (uuid, зовнішній ключ на AnalysisRequest.id), topics (json), keywords (json).

– сутність «Звіт з аналізу»: атрибути «Report» – report_id (uuid, первинний ключ), user_id (uuid, зовнішній ключ на User.id), analysis_id (uuid, зовнішній ключ на SentimentAnalysis.analysis_id), modeling_id (uuid, зовнішній ключ на TopicModeling.id), created_at (timestamp), content (blob).

– сутність «Посилання»: атрибути «ShareLinks» – id (uuid), report_id (uuid), url (text).

Сутність «User» представляє користувача системи, що має унікальний ідентифікатор (uuid), ім’я користувача та дату створення облікового запису. Це основна сутність, до якої прив’язуються всі операції аналізу тексту.

Сутність «AnalysisRequest» відповідає за запити користувачів на проведення аналізу текстових даних. Кожен запит має унікальний ідентифікатор, належить конкретному користувачеві, містить текстовий корпус для аналізу у форматі JSON і може мати зв’язок з попереднім запитом, якщо це продовження або модифікація раніше зробленого запиту. Також зберігається час, коли запит був зроблений.

Сутність «SentimentAnalysis» представляє результати аналізу настроїв для конкретного тексту, який входить до складу запиту. Вона містить унікальний ідентифікатор, прив’язку до запиту тексту та результат у вигляді JSON, де зберігаються дані про емоційне забарвлення тексту.

Сутність «TopicModeling» зберігає результати тематичного моделювання, що виконується на текстових даних. Вона також має унікальний ідентифікатор і прив’язку до конкретного запиту аналізу. У результатах моделювання зберігаються теми та ключові слова, які були знайдені під час аналізу, представлені у форматі JSON.

Сутність «Report» об’єднує результати аналізу настроїв та тематичного моделювання. Вона містить інформацію про те, який користувач створив звіт, а також посилання на результати аналізу настроїв і тем. Окрім цього, у звіті

зберігається дата створення та контент звіту у форматі BLOB (binary large object), який можна експортувати.

Сутність «ShareLink» дозволяє користувачу ділитися згенерованими звітами з іншими користувачами. Посилання прив’язане до конкретного звіту та має коротку URL-адресу для зручного доступу.

Для забезпечення максимальної ефективності схеми бази даних зроблено кілька важливих кроків. По-перше, для всіх сутностей використані унікальні ідентифікатори (UUID), що забезпечує високу масштабованість та дозволяє легко інтегрувати базу даних з іншими системами, незалежно від середовища або технологій. Це також зменшує ризик конфліктів при масштабуванні або паралельному виконанні операцій.

По-друге, для кожної сутності чітко визначені зовнішні ключі, що забезпечує збереження цілісності даних та зменшує можливість помилок при внесенні даних у базу. Зовнішні ключі також дозволяють швидко і точно знаходити відповідні записи між таблицями, що підвищує продуктивність запитів.

Також використовуються такі типи даних, як json і blob, що дозволяє ефективно зберігати текстові дані і результати аналізу без необхідності розбивати їх на дрібніші структури. Це значно спрощує управління складними даними та підвищує швидкість обробки великих обсягів тексту, наприклад, для збереження результатів аналізу настроїв і тематичного моделювання.

Оптимізація запитів була врахована шляхом нормалізації даних. Це дозволяє уникати дублювання інформації та зменшити обсяги збережуваних даних, а також підвищити швидкість читання і запису, оскільки таблиці стали легшими й компактнішими. Окремо винесені результати аналізу та звіти, що дозволяє зберігати великі обсяги даних без перевантаження основної структури.

Завдяки чіткій структурі зв’язків та добре продуманим індексам схема бази даних є не лише ефективною з точки зору зберігання, але й оптимізованою для

швидкого доступу до даних та мінімізації затримок під час виконання складних запитів.

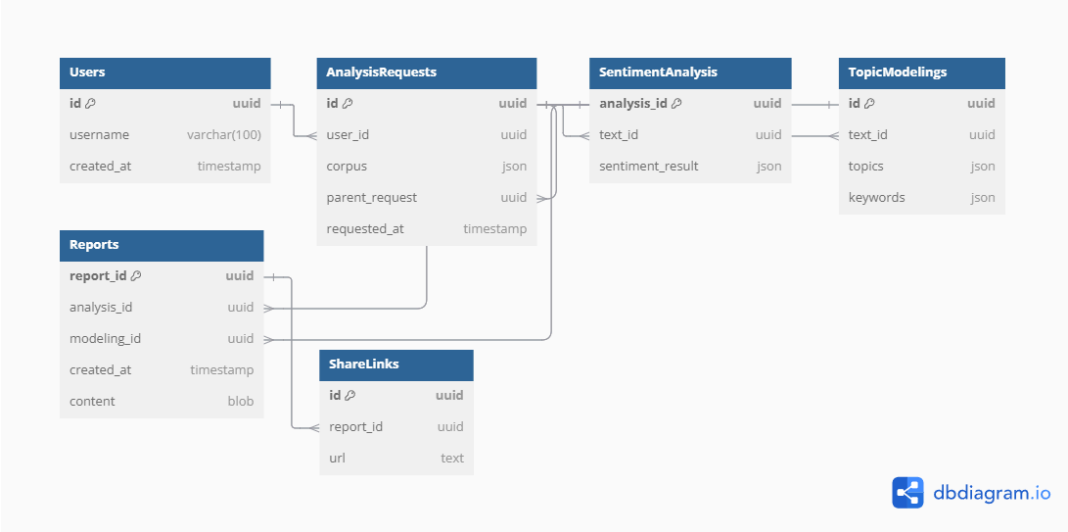

Рисунок 2.1 – ERD системи аналізу настроїв тексту на основі тематичного моделювання

Діаграма потоків даних (DFD) є важливим інструментом для візуалізації і аналізу процесів у системі, оскільки вона показує, як дані переміщуються між різними компонентами системи та як ці компоненти взаємодіють один з одним[14].

DFD рівня 0, або контекстна діаграма, представляє собою загальний огляд всієї системи або процесу, який підлягає аналізу або моделюванню. Ця діаграма слугує для швидкого сприйняття, демонструючи систему як єдиний високорівневий процес і відображаючи її взаємозв’язки із зовнішніми сутностями.

На діаграмі DFD 0 (рис. 2.2) продемонстровано загальний огляд основних процесів в системі.

Рисунок 2.2 – DFD рівня 0

DFD рівня 1 пропонує детальніший розгляд компонентів контекстної діаграми. Вона фокусується на основних функціях, які виконує система, розбиваючи загальний процес з контекстної діаграми на окремі підпроцеси. Ця діаграма дозволяє краще зрозуміти внутрішню структуру системи, розкриваючи її ключові етапи та взаємодії.

На діаграмі DFD 1 (рис. 2.3) продемонстровані основні підпроцеси системи, а саме аналіз настроїв та тематичне моделювання, включно з отриманням результатів для користувача.

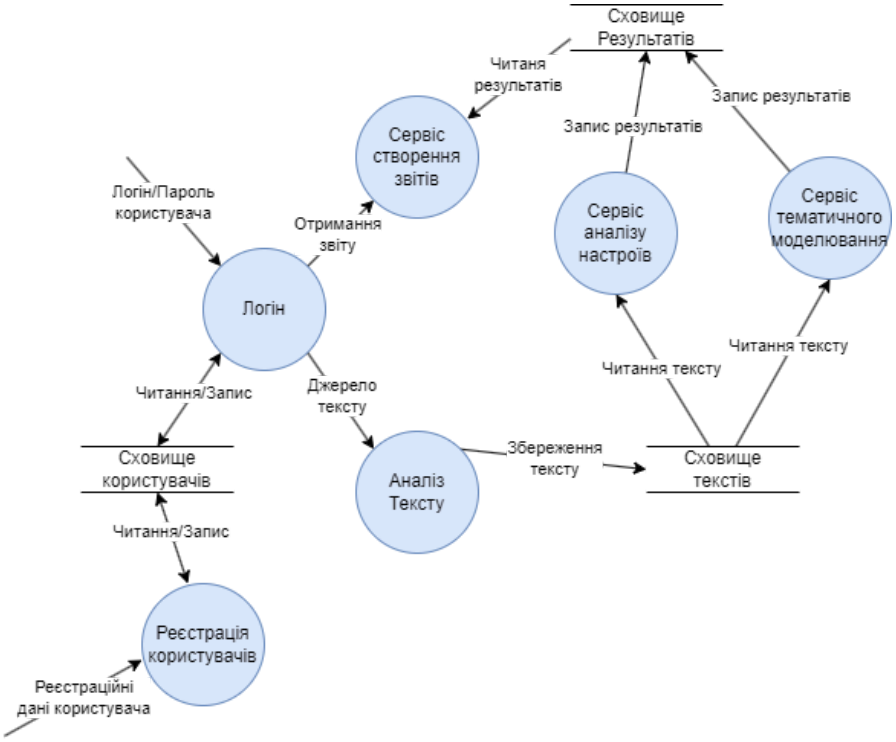

Рисунок 2.3 – DFD рівня 1

DFD рівня 2 робить ще один крок у поглибленому аналізі, деталізуючи частини, представлені на рівні 1.

На діаграмі DFD рівня 2 (рис. 2.4) зображено деталі процеси аналізу тексту, які включають в себе визначення типу джерела тексту, звернення до відповідного

парсера, процедури очищення та нормалізації тексту перед збереженням його у сховище.

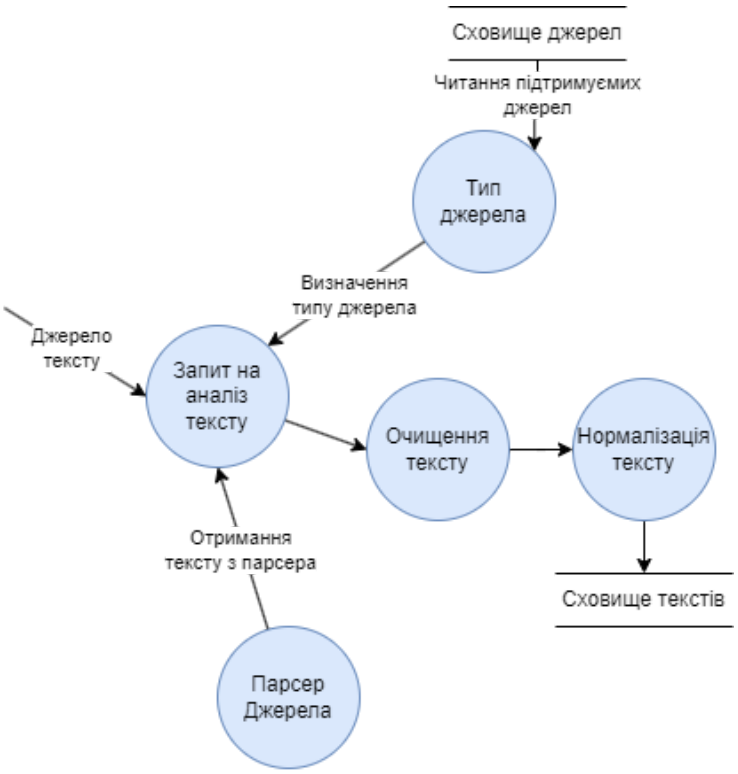

Рисунок 2.4 – DFD рівня 2

IDEF0 (Integration Definition for Function Modeling) – це методологія моделювання, яка використовується для опису функціональних аспектів системи [15]. Вона зосереджується на аналізі процесів та функцій, визначаючи, які вхідні дані використовуються, які ресурси потрібні, і які виходи генеруються в результаті виконання функцій. IDEF0 дозволяє розбити складну систему на ієрархію блоків для глибшого розуміння та управління кожним етапом.

Основна мета створення IDEF0 полягає у візуалізації та структуризації процесів системи. Завдяки IDEF0 можна наочно описати, як система взаємодіє з вхідними даними, які етапи обробки та процеси відбуваються всередині, і які результати генеруються. Це сприяє кращому плануванню архітектури системи, оптимізації функціональних вимог та зменшенню ризиків при її реалізації.

У контексті розробки системи аналізу настроїв тексту на основі тематичного моделювання діаграма IDEF0 може використовуватися для наступних цілей:

– аналіз і оптимізація процесів: IDEF0 допомагає чітко визначити всі ключові процеси, пов’язані з обробкою тексту та аналізом настроїв. Це дозволяє виявити можливі неефективності, дублювання або непотрібні кроки, що можуть сповільнювати роботу системи. Оптимізація цих процесів підвищить продуктивність системи та зменшить витрати ресурсів.

– планування архітектури системи: використання IDEF0 допомагає створити чітку структуру всіх функціональних компонентів системи. Це забезпечить правильне розуміння функціональності кожного елемента.

– узгодження вимог і функціональності: діаграма IDEF0 дозволяє чітко визначити функціональні вимоги системи та перевірити, чи відповідають ці вимоги початково заявленим цілям. Це дає можливість уникнути неправильного тлумачення завдань та забезпечити, щоб система на кожному етапі виконувала саме те, що від неї вимагається.

– взаємодія з користувачами та зовнішніми системами: IDEF0 дає змогу показати, як система аналізу настроїв буде взаємодіяти з іншими зовнішніми системами або користувачами, визначаючи входи та виходи процесів. Це важливо для інтеграції системи в робочі процеси, де вона може отримувати текстові дані з зовнішніх джерел і надавати результати аналізу користувачам або іншим системам.

– оцінка ресурсів і часу: IDEF0 також допомагає зрозуміти, які ресурси необхідні для виконання кожної функції (апаратні, програмні, людські). Це важливо для планування ресурсів, управління часом і бюджетом проєкту.

Блок A0 «Аналіз настроїв тексту» (рис. 2.5) є найвищим рівнем абстракції, що описує процес аналізу настроїв тексту в системі.

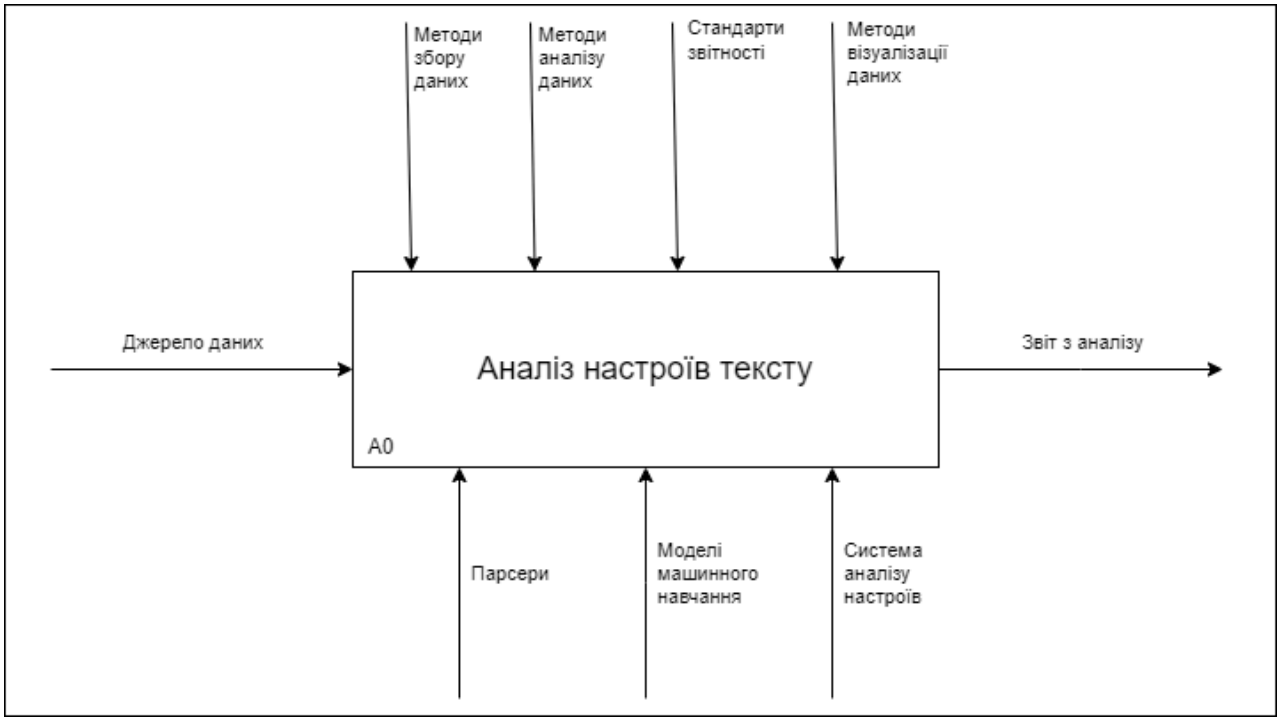

Рисунок 2.5 – Діаграма IDEF0, блок A0

Блок A0 складається з дочірніх підблокі, які використовує для виконання поставленої задачі. В свою чергу блоки A1-A4 (рис. 2.6) визначають основі етапи та функції з аналізу настроїв тексту.

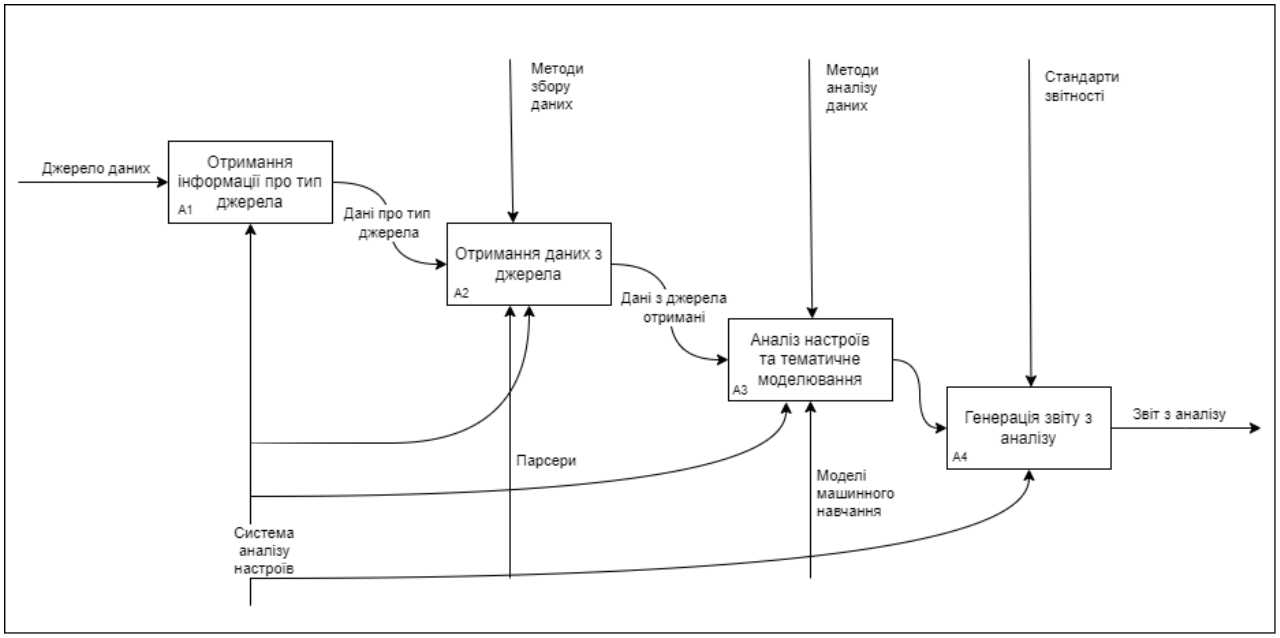

Рисунок 2.6 – Діаграма IDEF0, блок A1-A4

Діаграма IDEF0 з вузлами A1-A4 детально розкриває процес аналізу настроїв тексту. В табл. 2.1 наведено опис і функціонал кожного блоку, що дає змогу детально зрозуміти процеси в системі.

Таблиця 2.1 – Опис підблоків діграми IDEF0

Назва	**Опис**	**Функції**
A1: Отримання інформації про тип джерела	Процес визначення, з якого джерела буде отримано текстові дані (наприклад, веб, файл, API тощо).	Ідентифікація джерела даних, налаштування параметрів підключення до джерела.
A2: Отримання даних з джерела	Процес завантаження або отримання текстових даних з вказаного джерела.	Завантаження текстових даних, обробка отриманих даних.
A3: Аналіз настроїв та тематичне моделювання	Обробка тексту для визначення настрою (позитивного, негативного, нейтрального) та виділення тем.	Аналіз тексту на настрій, застосування тематичного моделювання для класифікації за ключовими темами.
A3: Аналіз настроїв та тематичне моделювання	Обробка тексту для визначення настрою (позитивного, негативного, нейтрального) та виділення тем.	Аналіз тексту на настрій, застосування тематичного моделювання для класифікації за ключовими темами.

Розроблена IDEF0 діаграма ефективно відображає всі основні функціональні етапи системи аналізу настроїв тексту на основі тематичного моделювання. Діаграма чітко структурує процеси від отримання інформації про джерела до генерації звітів, що дозволяє краще розуміти логіку функціонування системи, взаємодію її компонентів та послідовність операцій. Така структура дає можливість оптимізувати процеси на кожному етапі, ідентифікувати ресурси, необхідні для виконання кожної функції, і забезпечити ефективне управління даними.

Крім того, IDEF0 діаграма допомагає забезпечити прозорість у розробці, сприяє кращій комунікації між членами команди та покращує інтеграцію системи з іншими компонентами, зокрема щодо отримання та обробки текстових даних. Таким чином, діаграма є важливим інструментом для забезпечення якості та узгодженості системи, яка розробляється.

2.2 Моделювання сценаріїв використання

Use Case – це опис специфічної взаємодії користувача (актора) з системою для досягнення певної цілі. Use Case описує дії або кроки, які користувач повинен виконати для взаємодії з системою, а також реакцію системи на ці дії. Це інструмент для визначення вимог до системи, який допомагає зрозуміти, як система буде використовуватися кінцевими користувачами.

Use Case Diagram – це графічне представлення взаємодії користувачів із системою. На діаграмі показані актори (користувачі або інші системи) і випадки використання (дії, які вони виконують у системі). Діаграма допомагає візуалізувати функціональні вимоги до системи та продемонструвати, які процеси відбуваються в ній з точки зору користувача [16]. Вона використовується для ілюстрації ролей користувачів та їх взаємодій із системою, полегшуючи комунікацію між розробниками, аналітиками та зацікавленими сторонами.

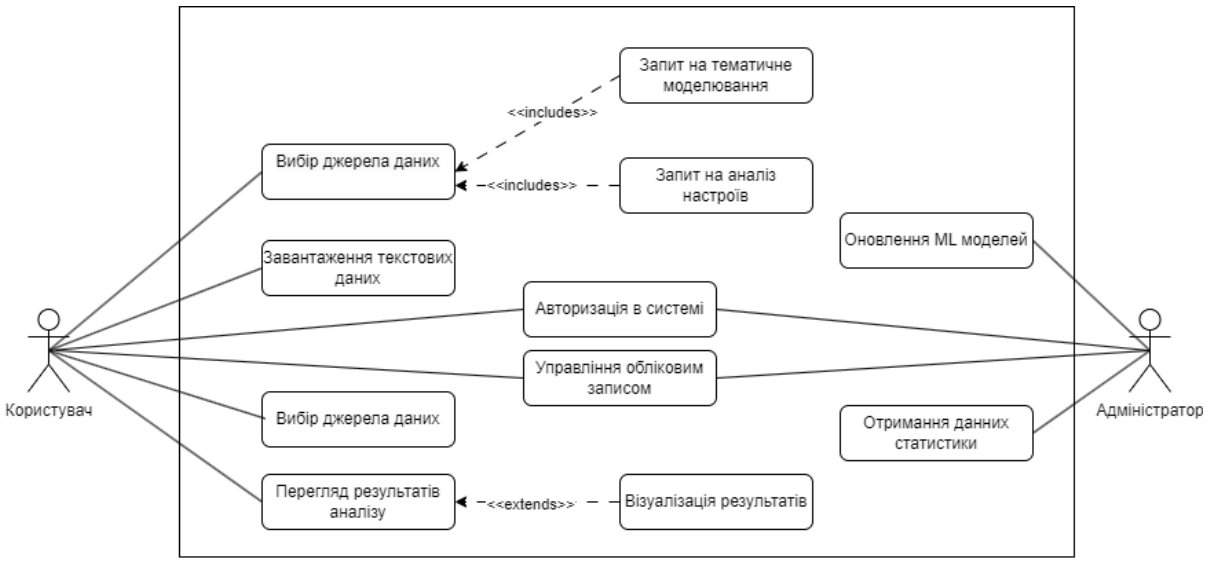

Рисунок 2.7 – Use case діаграма системи, що розробляється

Короткий Use Case:

Користувач, вибравши джерело даних, завантажує текстові дані в систему. Після цього користувач ініціює запит на аналіз настроїв, обираючи відповідну функцію на інтерфейсі системи. Система отримує текст і передає його на обробку, виконуючи аналіз настроїв з використанням машинного навчання, що визначає емоційну забарвленість тексту (позитивну, негативну або нейтральну). Під час цього процесу система викликає підпроцес аналізу тексту, де кожен фрагмент перевіряється на настрій. Після завершення обробки результати зберігаються в базі даних, і користувач може переглянути результати у вигляді візуалізованих даних у своєму профілі.

Поверхневий Use Case:

Головний сценарій (успішний):

1. користувач вибирає джерело даних;
2. завантажує текстові дані до системи;
3. ініціює запит на аналіз настроїв через інтерфейс системи;
4. система обробляє текст, виконує аналіз настроїв;

5. після завершення аналізу система зберігає результати;

6. користувач переглядає результати візуалізованого аналізу в своєму профілі.

Альтернативний сценарій (неуспішний):

1. користувач вибирає джерело даних;

2. завантажує текстові дані, але система стикається з помилкою під час обробки;

3. система відображає повідомлення про помилку в інтерфейсі;

4. користувач може спробувати перезавантажити дані або звернутися до технічної підтримки.

Таблиця 2.2 – Повний usecase

Розділ	**Опис**
Use Case Name	Запит на аналіз настроїв
Scope	Система аналізу настроїв тексту на основі тематичного моделювання
Level	Користувацький рівень
Primary Actor	Користувач
Stakeholders and Interests	– Звичайний користувач: Зацікавлений в отриманні швидкого та точного аналізу настроїв тексту для власних досліджень або бізнес-цілей. – Адміністратор: Зацікавлений у забезпеченні стабільної роботи системи та усуненні помилок під час аналізу.

Продовження таблиці 2.2

Розділ	**Опис**
Preconditions	– Користувач зареєстрований у системі та авторизований. – Текстові дані завантажені в систему або є доступ до відповідного джерела даних.
Success Guarantee	Система надає користувачу результати аналізу настроїв тексту та тематичного моделювання без помилок, забезпечуючи зручний інтерфейс для перегляду і подальшої роботи з даними.
Main Success Scenario	Користувач: – вибирає джерело текстових даних через інтерфейс системи; – завантажує текстові дані або підключається до існуючого джерела; – ініціює запит на аналіз настроїв. Система: – обробляє дані за допомогою алгоритмів машинного навчання та аналізує настрій тексту. – після завершення аналізу система генерує результати у вигляді візуалізованих даних. – користувач переглядає результати через інтерфейс і, за потреби, експортує звіт.

Кінець таблиці 2.2

Розділ	**Опис**
Extensions	– Помилка під час завантаження тексту: Якщо завантажені дані некоректні або пошкоджені, система відображає повідомлення про помилку та пропонує завантажити інший файл або повторити спробу.
Special Requirements	– Висока точність аналізу настроїв (не менш ніж 90% точності). – Час обробки тексту повинен бути мінімізований, щоб результати надавалися швидко, навіть при великому обсязі даних.
Special Requirements	– Висока точність аналізу настроїв (не менш ніж 90% точності). – Час обробки тексту повинен бути мінімізований, щоб результати надавалися швидко, навіть при великому обсязі даних.
Technology and Data	– Технології: Машинне навчання для аналізу настроїв, алгоритми тематичного моделювання (наприклад, LDA), база даних для збереження результатів. – Дані: Вхідні текстові дані у форматах JSON, CSV, або plain text, результати аналізу у вигляді JSON-об’єктів.
Variations List	Підтримка різних джерел: API, файли, інші платформи
Frequency of Occurrence	Регулярно

Створені сценарії взаємодії з системою чітко визначають, які шляхи взаємодії будуть у користувача з системою.

2.3 Алгоритм роботи програмного забезпечення

Алгоритм роботи програми – це чітка, покрокова послідовність дій або інструкцій, яку виконує програма для досягнення конкретної мети. Алгоритм включає всі етапи від введення даних, їх обробки до виведення результатів. Це фундаментальна частина програмного забезпечення, яка визначає, як система взаємодіє з користувачем, обробляє дані і виконує свої функції.

Моделювання алгоритмів у контексті розробки системи аналізу настроїв тексту на основі тематичного моделювання є важливим, оскільки воно дозволяє визначити ефективні шляхи для обробки великих обсягів даних, їх аналізу та виведення результатів [17]. Правильне моделювання допомагає оптимізувати виконання завдань, мінімізувати помилки та забезпечити швидку роботу програми, що особливо важливо при обробці великих об’ємів даних. Це також дає змогу розробникам зрозуміти процеси та етапи роботи програми на ранніх стадіях, уникнути неефективних рішень і врахувати потенційні проблеми ще до впровадження системи.

У контексті цієї системи, що розробляється, алгоритми відіграють особливу роль в автоматизації процесу аналізу (рис. 2.8), оскільки вони дозволяють розбити текст на сегменти, виявити приховані теми та емоційні забарвлення тексту. Алгоритми машинного навчання, які застосовуються для аналізу настроїв і тематичного моделювання, не тільки обробляють великі обсяги інформації, але й адаптуються до нових даних, що дозволяє системі залишатися актуальною і точнішою з часом.

Крім того, чітко спроєктовані алгоритми допомагають мінімізувати час обробки тексту, що критично важливо для користувачів, яким необхідні результати

в режимі реального часу або близько до нього. Надійні алгоритми також забезпечують надійність і стабільність системи, адже чим краще розроблені алгоритми, тим менше ймовірність виникнення помилок або збоїв під час виконання аналізу.

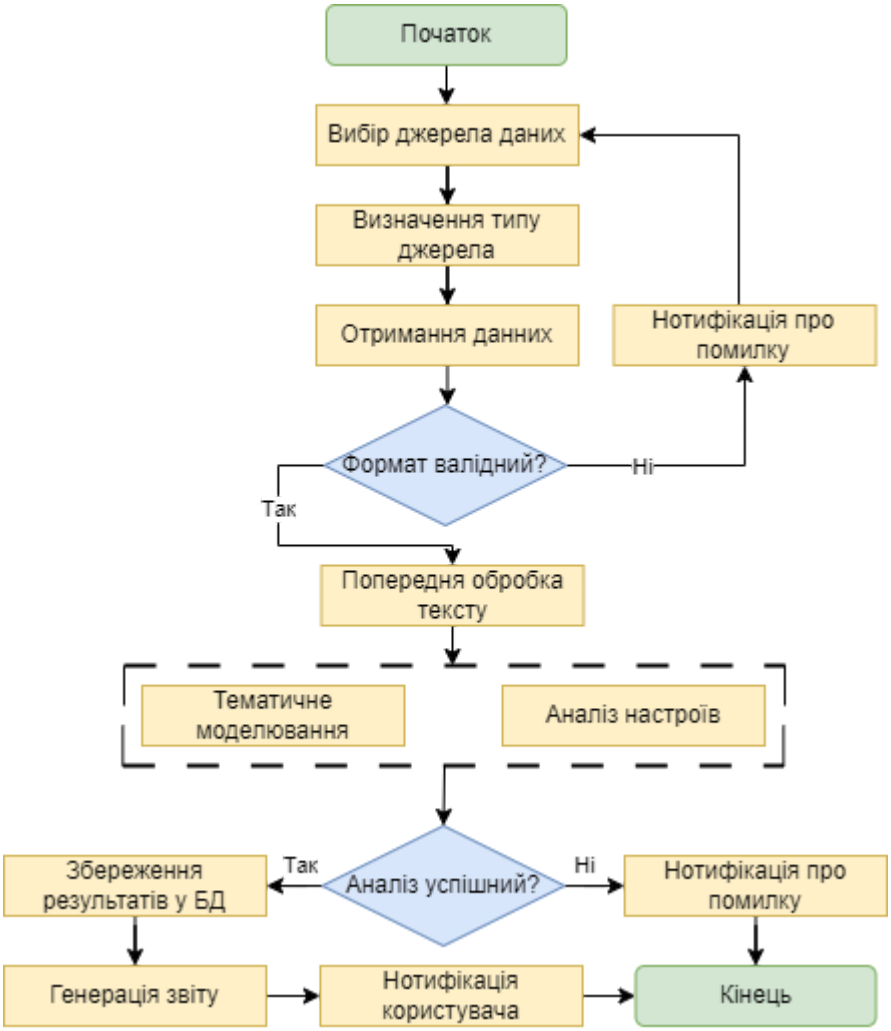

Рисунок 2.8 – Блок-схема алгоритму аналізу настроїв

Блок-схема описує основний алгоритм аналізу тексту:

1. **Вибір джерела даних** – користувач починає роботу із системою, вибираючи джерело даних для аналізу. Це може бути як завантажений файл, так і підключення до зовнішнього API. Важливо, що на цьому етапі відбувається інтеракція з

інтерфейсом системи, де користувач надає або визначає шлях до джерела інформації.

2. **Визначення типу джерела** – після того як користувач обрав джерело даних, система визначає, чи це локальний файл, чи зовнішній ресурс (API). Цей етап критичний для подальшої обробки, оскільки залежно від типу джерела даних застосовуються різні механізми отримання та перевірки інформації.

3. **Отримання даних** – в залежності від вибору, дані або завантажуються з локального пристрою, або отримуються через API-запит. Система забезпечує правильне отримання даних і перевірку на наявність доступу або помилок при запиті до API.

4. **Перевірка формату** – отримані дані проходять перевірку на відповідність підтримуваним форматам (наприклад, .txt, .csv, .json). Якщо дані не відповідають вимогам, користувач отримує повідомлення про помилку та пропозицію надати нове джерело даних.

5. **Попередня обробка тексту** – на цьому етапі система готує текст для подальшого аналізу. Це включає токенізацію, видалення стоп-слів, лемматизацію або стемінг, та інші операції, що покращують якість даних для наступних етапів.

6. **Тематичне моделювання та аналіз настроїв** – система виконує два ключові завдання: проводить тематичне моделювання тексту, що дозволяє виявити ключові теми, та аналізує настрій тексту, класифікуючи його як позитивний, негативний або нейтральний. Це основний етап аналізу, який потребує машинного навчання та обробки великих обсягів тексту.

7. У разі **успішного аналізу** результати зберігаються у базі даних, що дозволяє їх використовувати для подальших досліджень або звітів. Система автоматично формує звіт на основі отриманих результатів, який користувач може експортувати в форматі PDF. Користувач отримує повідомлення про завершення аналізу та можливість перегляду або збереження результатів.

8. У випадку **невдалої спроби аналізу**, система повідомляє користувача про помилку та пропонує повторно виконати операцію або перевірити надані дані.

2.4 Створення діаграми розгортання

Діаграма розгортання (Deployment Diagram) — це тип діаграми в моделюванні UML (Unified Modeling Language), яка показує фізичну архітектуру системи [18]. Вона демонструє, як програмні компоненти системи розгорнуті на апаратних засобах (сервери, комп'ютери, мобільні пристрої тощо) та як ці елементи взаємодіють між собою. Діаграма відображає вузли (nodes), на яких виконуються окремі компоненти програми, і канали зв'язку між ними.

Мета створення діаграми розгортання полягає в наданні чіткої та зрозумілої візуалізації фізичної архітектури програмної системи, що дозволяє розробникам, архітекторам і зацікавленим сторонам зрозуміти, як компоненти системи розгорнуті на різних апаратних засобах. Це важливо для оцінки ефективності, безпеки та надійності рішення. Діаграма допомагає ідентифікувати потенційні проблеми, такі як перевантаження серверів, недостатня потужність обладнання або неефективна взаємодія між компонентами.

На діаграмі розгортання (рис. 2.9) зображені наступні елементи:

1. **LoadBalancer/Gateway** – це компонент, який розподіляє вхідні запити між різними серверами, забезпечуючи балансування навантаження та підвищуючи доступність системи. Він приймає запити від користувачів і передає їх на відповідні сервери, що дозволяє уникнути перевантаження окремих серверів і забезпечити швидкий доступ до ресурсів.

2. **Frontend App, Backend App, ML Service** – цей сервер містить як фронтенд, бекенд і ML застосуноки. Фронтенд відповідає за інтерфейс користувача, забезпечуючи взаємодію з клієнтами через вебзастосунок, тоді як

бекенд обробляє бізнес-логіку, керує запитами до бази даних, а ML сервіс виконує необхідні обчислення для аналізу даних.

3. **DB Server** – це основний сервер, що містить розгорнуту базу даних, Цей сервер зазвичай обробляє всі операції запису. Сервер також містить репліку основної бази даних і призначений для обробки запитів на читання. Це дозволяє зменшити навантаження на основну базу даних.

4. **Broker Server** – це сервер, що використовує Rabbit MQ як брокер повідомлень для асинхронної комунікації між компонентами системи. Він дозволяє різним частинам програми обмінюватися даними без прямої залежності один від одного, що підвищує масштабованість та надійність системи. Цей компонент особливо корисний для обробки великих обсягів даних і управління чергами завдань.

5. Auth Server – це сервер, що містить розгорнутий екземпляр keycloak, який забезпечує авторизацію та авентифікацію користувачів в системі. Даний компонент є критично важливим для роботи системи, тож потребує окремого сервера для розгортання.

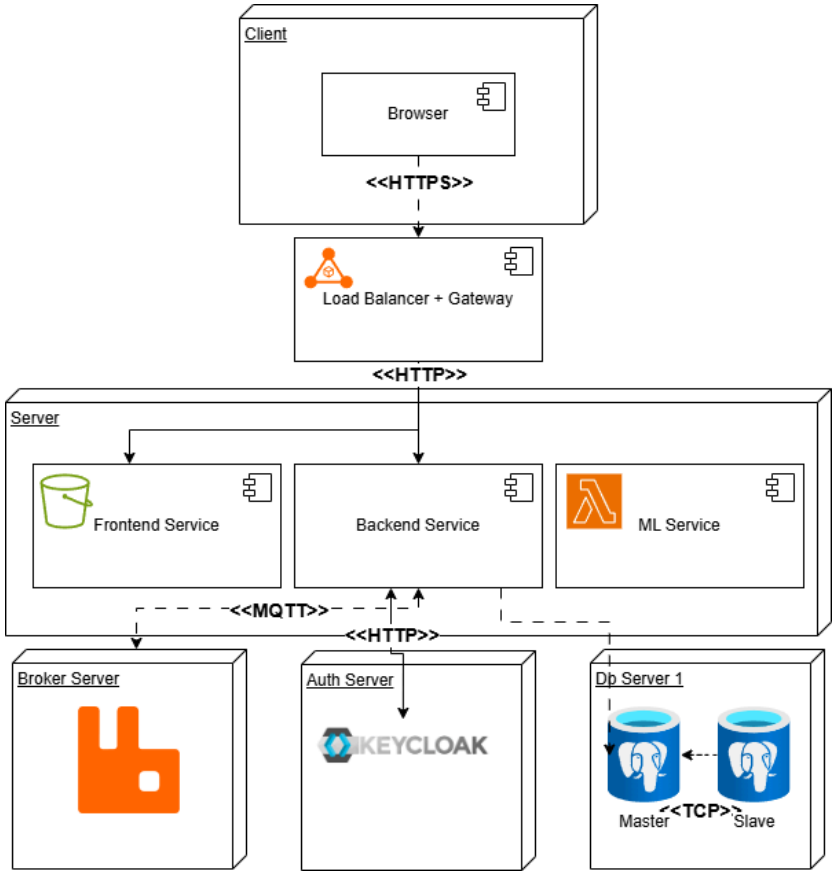

Рисунок 2.9 – Діаграма розгортання системи

Діаграма розгортання надає чітке уявлення про фізичну архітектуру системи аналізу настроїв тексту на основі тематичного моделювання. Вона ілюструє, як основні компоненти, такі як LoadBalancer, сервери фронтенду та бекенду, репліки бази даних і брокер повідомлень, взаємодіють між собою для забезпечення ефективної обробки запитів і зберігання даних.

Висновки до розділу 2

Перший етап розділу включала в себе створення моделей системи, таких як ERD (Entity Relationship Diagram) та DFD (Data Flow Diagram) на рівнях 0-2, а також IDEF0. Ці діаграми дозволили чітко визначити структуру даних, їхні взаємозв’язки та потоки інформації, що забезпечує ефективне проєктування бази даних та механізмів обробки даних. Використання цих моделей сприяло

розумінню логіки функціонування системи та допомогло виявити можливі проблеми на етапі проєктування.

Другий етап включав створення діаграми сценаріїв використання (Use Case Diagram) та опис основних варіантів використання системи. Це дозволило визначити ключові функції системи з точки зору користувача, а також встановити зв’язок між різними акторами та їхніми взаємодіями з системою. Описані варіанти використання допомогли зосередити увагу на основних потребах користувачів і забезпечили основу для подальшої розробки функціональних вимог.

Наступним кроком була розробка блок-схеми алгоритму роботи системи, яка представила покрокову логіку виконання основних функцій. Блок-схема дала можливість візуально зрозуміти, як дані проходять через систему, а також вказала на критичні етапи, які потребують особливої уваги під час реалізації. Це також слугувало основою для виявлення можливих оптимізацій у процесі обробки даних.

Останній етап – створення діаграми розгортання, яка демонструє фізичну архітектуру системи та її компоненти. Ця діаграма важлива для розуміння, як програмні модулі взаємодіють з апаратним забезпеченням і як організовані їхні зв’язки. Вона також дозволила оцінити продуктивність системи та спростила процес планування ресурсів.

У загальному підсумку, проведений аналіз і моделювання системи допомогли не лише візуалізувати її структуру і функціональність, але й визначити ключові аспекти для подальшої розробки. Всі створені діаграми забезпечують цілісне розуміння системи, що дозволяє знизити ризики на етапі реалізації та підвищити загальну ефективність проєкту. Цей структурований підхід закладає основи для успішної реалізації системи, враховуючи потреби користувачів і технологічні вимоги.

3 ПРОЄКТУВАНЯ СИСТЕМИ ТА ОГЛЯД ТЕХНОЛОГІЧНОГО СТЕКУ

3.1 Створення UML-діаграм

UML (Unified Modeling Language) діаграми – це стандартний спосіб візуалізації дизайну та структури програмних систем. Вони допомагають розробникам, архітекторам та іншим зацікавленим сторонам зрозуміти, як різні елементи системи взаємодіють між собою, а також дозволяють моделювати складні програмні системи на різних рівнях абстракції [19].

UML-діаграми дозволяють описати структуру та поведінку програмного забезпечення через використання різних типів діаграм, які фокусуються на певних аспектах системи, таких як класи, компоненти, потоки даних або взаємодії.

Основні діаграми, що є корисними під час проєктування системи: діаграми класів, компонентів та пакетів.

Діаграма класів є однією з найважливіших UML-діаграм. Вона описує структуру системи через відображення її класів, їхніх атрибутів, методів і відносин між класами, таких як наслідування, асоціації або агрегації [19]. Діаграми класів допомагають розуміти об’єктно-орієнтовану структуру системи та відображають ієрархії класів, їх залежності і способи взаємодії.

Діаграма компонентів показує, як різні програмні компоненти взаємодіють між собою. Вона відображає модульність системи, зображуючи компоненти, їхні залежності та з’єднання [19]. Це дозволяє розробникам зрозуміти, як різні частини системи об’єднуються в єдине ціле і як дані передаються між компонентами.

Діаграма пакетів показує, як різні логічні блоки або пакети системи організовані і пов’язані один з одним. Вона ілюструє розподіл програмного коду на модулі, групи класів та компоненти [19]. Це дозволяє краще організувати структуру програми, полегшує навігацію та підтримку коду. Діаграма пакетів

особливо корисна при великих і складних проєктах, оскільки допомагає тримати структуру системи упорядкованою та легко масштабованою.

3.1.1 Діаграма класів

У контексті розробки системи аналізу настроїв тексту, діаграма класів (рис. 3.1) представляє класи, що відповідають за текстовий аналіз, обробку даних, роботу з базою даних та генерацію звітів.

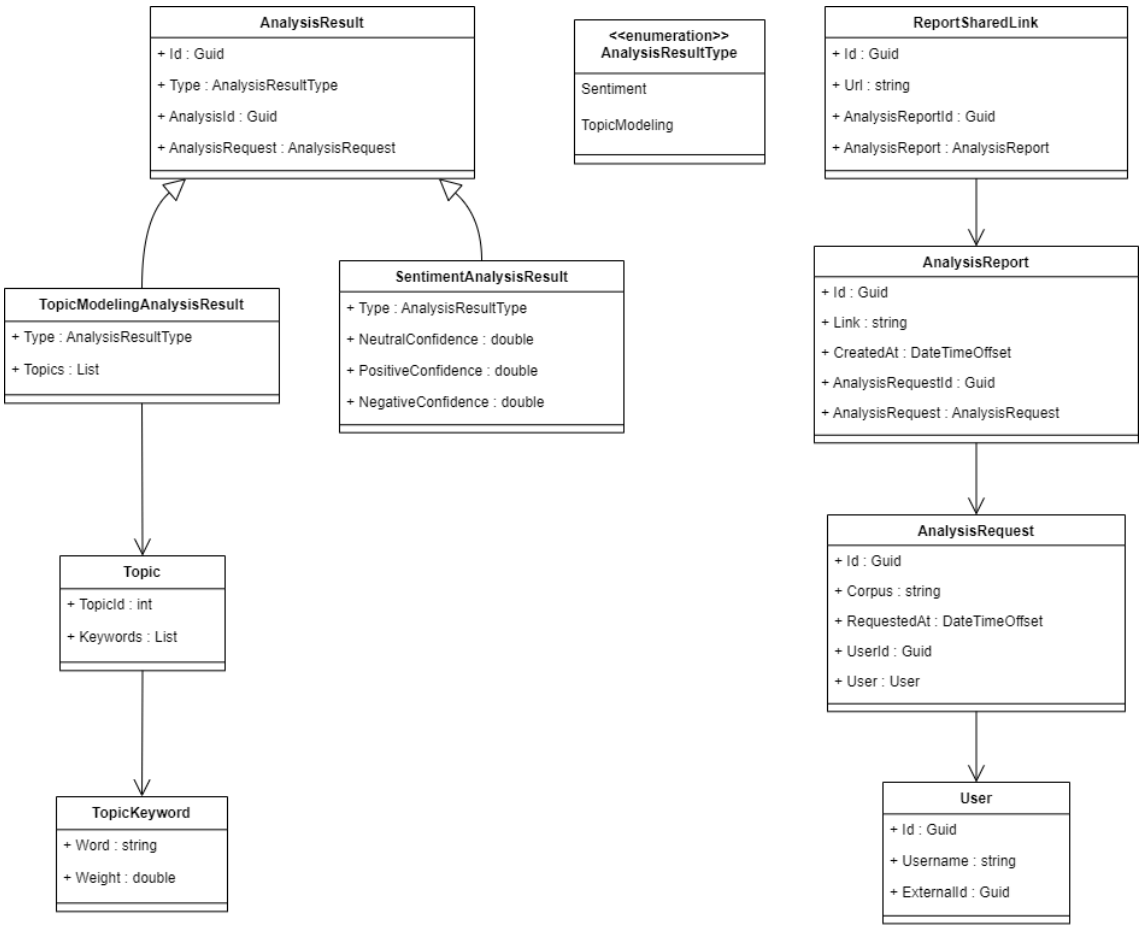

Рисунок 3.1 – Діаграма класів системи

Діаграма класів складається з наступних класів:

– AnalysisReport: представляє звіт про аналіз, містить посилання та інформацію про запит аналізу;

– AnalysisRequest: містить дані про запит на аналіз тексту, включаючи текст і користувача, що його створив;

– AnalysisResult: представляє результат аналізу, який може бути типу «аналіз настроїв» або «тематичне моделювання»;

– AnalysisResultType: перелік типів аналізу, доступних у системі (аналіз настроїв або тематичне моделювання);

– ReportSharedLink: містить інформацію про посилання для спільного доступу до звіту;

– SentimentAnalysisResult: результат аналізу настроїв з показниками впевненості для нейтрального, позитивного і негативного настроїв;

– TopicModelingAnalysisResult: результат тематичного моделювання, що містить список тем;

– Topic: представляє одну тему, що складається з ключових слів;

– TopicKeyword: представляє ключове слово і його вагу для певної теми;

– User: представляє користувача, що ініціював запит на аналіз.

Таблиця 3.1 – Класи та їх атрибути

Клас	**Атрибути**
AnalysisReport	Id, Link, CreatedAt, AnalysisRequestId, AnalysisRequest
AnalysisRequest	Id, Corpus, RequestedAt, UserId, User
AnalysisResult	Id, Type, AnalysisId, AnalysisRequest
AnalysisResultType	Sentiment, TopicModeling
ReportSharedLink	Id, Url, AnalysisReportId, AnalysisReport
SentimentAnalysisResult	Type, NeutralConfidence, PositiveConfidence, NegativeConfidence

Кінець таблиці 3.1

Клас	Атрибути
TopicModelingAnalysisResult	Type, Topics
Topic	TopicId, Keywords
TopicKeyword	Word, Weight
User	Id, Username, ExternalId

Створені діаграми класів будуть основою під час проєктування системи і допоможуть уникнути неправильного трактування меж відповідальностей сутностей.

3.1.2 Діаграма проєктів

Діаграма проєктів у C#-системі є візуальним відображенням організації модулів та їхніх залежностей. Вона показує, як різні частини коду взаємодіють між собою, розкриває структуру системи та дозволяє зрозуміти, які компоненти відповідають за бізнес-логіку, доступ до бази даних, взаємодію з зовнішніми системами та інші функціональні частини.

Система складається з кількох ключових компонентів (рис. 3.2), що виконують свої спеціалізовані функції. Один із них – це TextAnalyzer.Core, який виступає як центральний модуль з базовими моделями та інтерфейсами. Він надає фундаментальні класи та типи, що використовуються у всіх інших модулях системи. Завдяки цьому TextAnalyzer.Core служить базою для розширення функціоналу та реалізації нових можливостей.

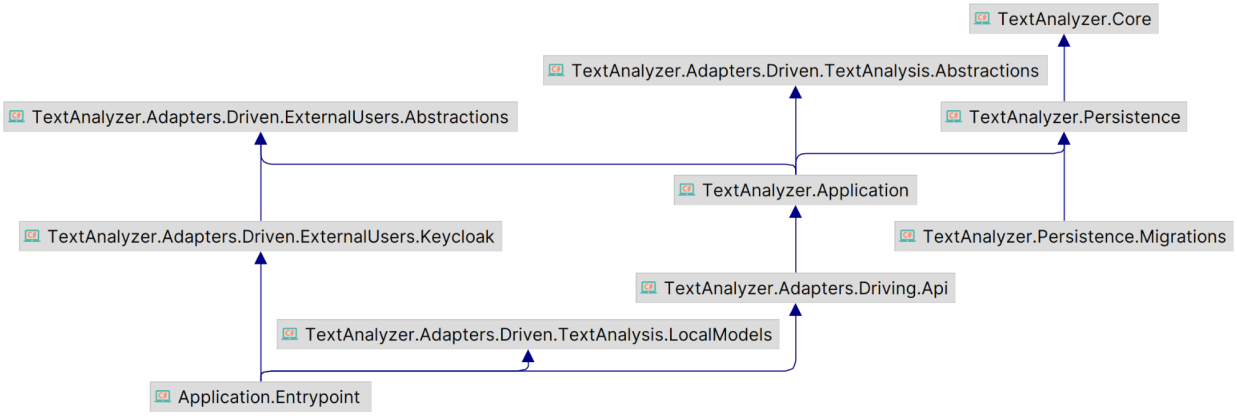

Рисунок 3.2 – Діаграма проєктів

Інший важливий модуль – TextAnalyzer.Application, який реалізує бізнес-логіку системи. Він координує взаємодію між різними компонентами, забезпечуючи виконання основних функцій системи, таких як аналіз тексту, обробка запитів користувачів та управління даними. Цей модуль використовує ресурси TextAnalyzer.Core для роботи з базовими моделями та забезпечує зв’язок із зовнішніми API та адаптерами.

Для взаємодії системи з зовнішніми клієнтами використовується модуль TextAnalyzer.Adapters.Driving.Api. Він відповідає за надання API, через який зовнішні програми та сервіси можуть отримувати доступ до функціоналу аналізу тексту. Цей модуль отримує дані від TextAnalyzer.Application та надає їх у зручному форматі для зовнішніх користувачів, забезпечуючи інтеграцію з іншими системами.

Система також включає модуль *TextAnalyzer.Adapters.Driven.TextAnalysis.LocalModels*, який містить реалізації моделей для локального аналізу тексту. Цей модуль відповідає за конкретну реалізацію алгоритмів аналізу та обробки тексту, що використовуються системою. Він взаємодіє з TextAnalyzer.Application, забезпечуючи інструменти для аналізу настроїв та тематичного моделювання на основі локально збережених моделей.

Для підтримки гнучкості та можливості зміни реалізацій в системі передбачено модуль *TextAnalyzer.Adapters.Driven.TextAnalysis.Abstractions*. Він містить інтерфейси та абстракції, які визначають, як система повинна взаємодіяти з модулями аналізу тексту. Це дозволяє розробникам легко замінювати конкретні моделі або адаптери, не змінюючи основний код бізнес-логіки.

Зберігання даних у системі здійснюється через модуль *TextAnalyzer.Persistence*. Він відповідає за роботу з базою даних, зберігає результати аналізу та іншу необхідну інформацію. TextAnalyzer.Persistence використовує моделі та інтерфейси з *TextAnalyzer.Core* для роботи з даними, а також інтегрується з *TextAnalyzer.Application* для збереження результатів обробки тексту.

Для управління змінами у структурі бази даних система має окремий модуль *TextAnalyzer.Persistence.Migrations*. Він містить скрипти міграцій та забезпечує автоматичне оновлення схеми бази даних, що спрощує підтримку та розвиток системи. Це дозволяє безпечно додавати нові таблиці чи змінювати структуру існуючих даних.

Для роботи з зовнішніми системами аутентифікації користувачів у системі реалізовано модуль *TextAnalyzer.Adapters.Driven.ExternalUsers.Abstractions*. Він містить інтерфейси, що визначають взаємодію з різними провайдерами аутентифікації. Це забезпечує підтримку різних платформ, не залежачи від конкретної реалізації.

Конкретна реалізація інтеграції з Keycloak, популярною системою для управління користувачами та авторизацією, представлена в модулі *TextAnalyzer.Adapters.Driven.ExternalUsers.Keycloak*. Цей модуль реалізує інтерфейси з *TextAnalyzer.Adapters.Driven.ExternalUsers.Abstractions* та надає функції для аутентифікації користувачів через Keycloak, дозволяючи використовувати цей сервіс для авторизації в системі.

Application.EntryPoint – це точка входу в застосунок, де відбувається ініціалізація всіх сервісів та модулів. Тут здійснюється налаштування взаємодії між модулями, запуск API, підключення до бази даних та інші процеси, необхідні для коректної роботи системи. Це забезпечує зручний старт системи та гнучкість у її налаштуванні.

Ця архітектура побудована на принципі поділу на абстракції та конкретні реалізації, що дозволяє легко змінювати окремі компоненти без впливу на основну бізнес-логіку [29]. Це забезпечує можливість масштабування системи, покращує її підтримку та дає змогу швидко адаптуватися до нових вимог, зберігаючи високу якість коду та структуру проєкту.

3.2 Проєктування інтерфейсу системи

Проєктування інтерфейсу користувача є одним з ключових етапів створення сучасного програмного забезпечення. Інтерфейс відіграє важливу роль у забезпеченні зручності та ефективності взаємодії користувача з системою. Від якості інтерфейсу залежить не лише задоволеність користувача, але й загальна продуктивність роботи з програмою, адже інтуїтивно зрозумілий і логічний інтерфейс дозволяє значно скоротити час на виконання основних дій та зменшити кількість можливих помилок.

Проєктування інтерфейсу системи аналізу настроїв тексту на основі тематичного моделювання є важливим, оскільки користувачі мають легко орієнтуватися в результатах аналізу та зручно взаємодіяти з функціями системи, такими як створення звітів, перегляд звітів аналізу тощо. Інтерфейс має забезпечувати простоту доступу до кожної з цих функцій та не обтяжувати користувача зайвими елементами, зберігаючи баланс між інформаційною насиченістю та простотою використання.

У процесі роботи були створені макети таких сторінок: логін, реєстрація, перегляд аналізів тексту, створення запиту на аналіз та перегляд звіту аналізу. Кожна з цих сторінок виконує свою конкретну роль в забезпеченні логічного та зручного інтерфейсу для користувачів, що взаємодіють із системою аналізу настроїв тексту на основі тематичного моделювання.

Сторінка авторизації (рис. 3.3) забезпечує вхід до системи, надаючи користувачам простий доступ до свого акаунту, де вони можуть переглядати попередні результати аналізів та створювати нові запити. Дизайн цієї сторінки орієнтований на швидкість і простоту, що дозволяє користувачам зручно вводити свої облікові дані та здійснювати доступ до системи.

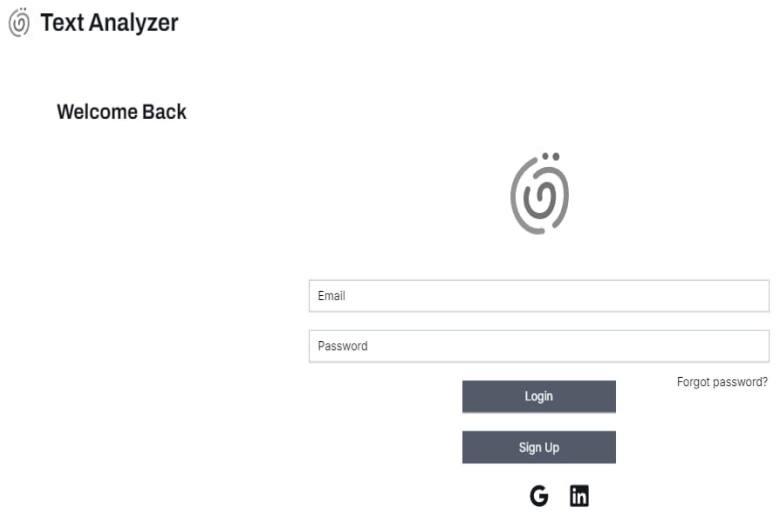

Рисунок 3.3 – Дизайн сторінки авторизації

Сторінка реєстрації (рис. 3.4) призначена для створення нових облікових записів. Її структура передбачає поля для введення основних даних, таких як ім’я користувача, електронна пошта та пароль. Особливу увагу приділено валідації введених даних, щоб забезпечити точність та безпеку під час реєстрації нових користувачів. Простий і зрозумілий інтерфейс цієї сторінки допомагає користувачам швидко розпочати роботу з системою.

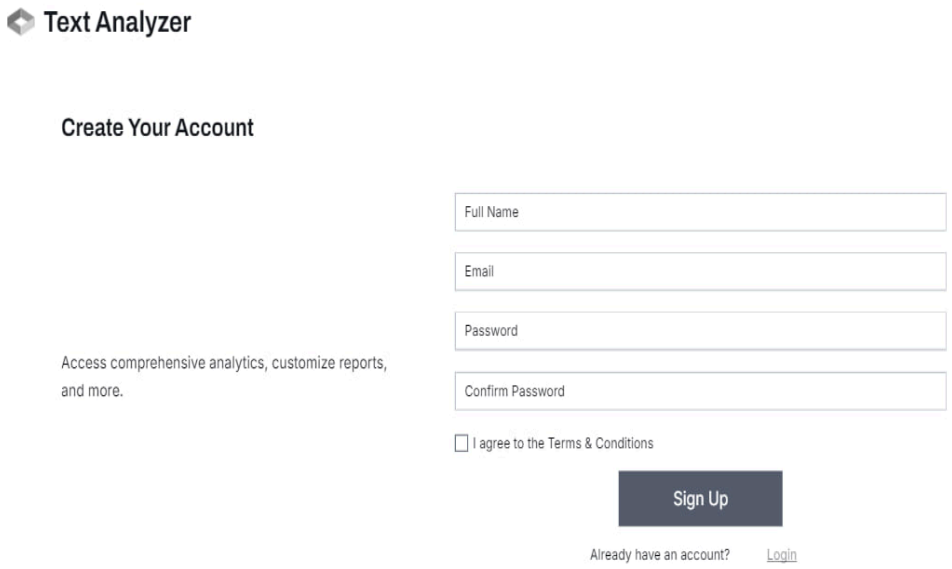

Рисунок 3.4 – Дизайн сторінки реєстрації

Сторінка створення запиту на аналіз (рис. 3.5) дозволяє користувачам подавати тексти на обробку системою. Тут користувач може вибрати тип аналізу, ввести або завантажити текст для обробки, а також налаштувати необхідні параметри. Інтерфейс цієї сторінки спрощує процес налаштування та подачі запиту, забезпечуючи гнучкість і зручність для різних типів користувачів.

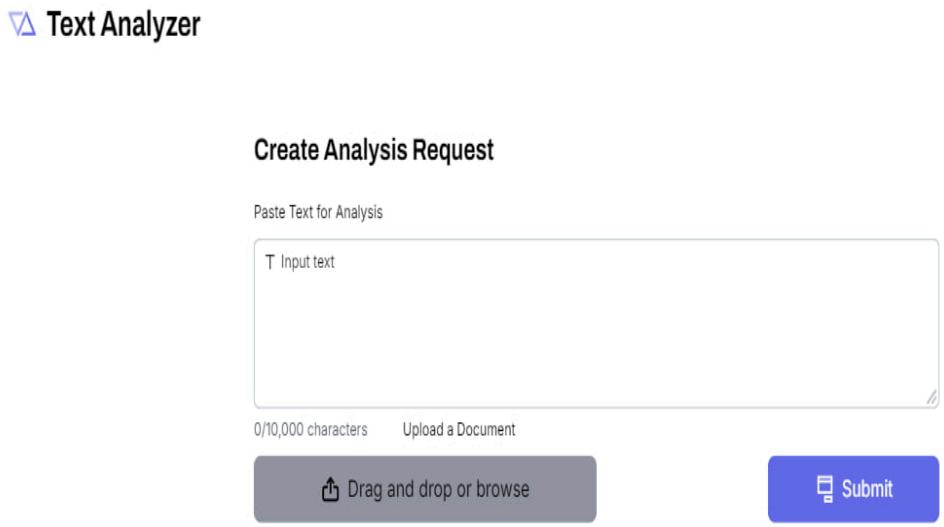

Рисунок 3.5 – Дизайн сторінки створення запиту на аналіз

Сторінка перегляду аналізів тексту (рис. 3.6) надає користувачам доступ до результатів раніше виконаних запитів на аналіз. Вона містить таблицю з переліком аналізів, що включає основні деталі, такі як дата створення, статус та тип аналізу. Це дає можливість швидко знайти потрібний результат і переглянути його деталі. Для зручності користувача реалізовано фільтрацію та сортування результатів.

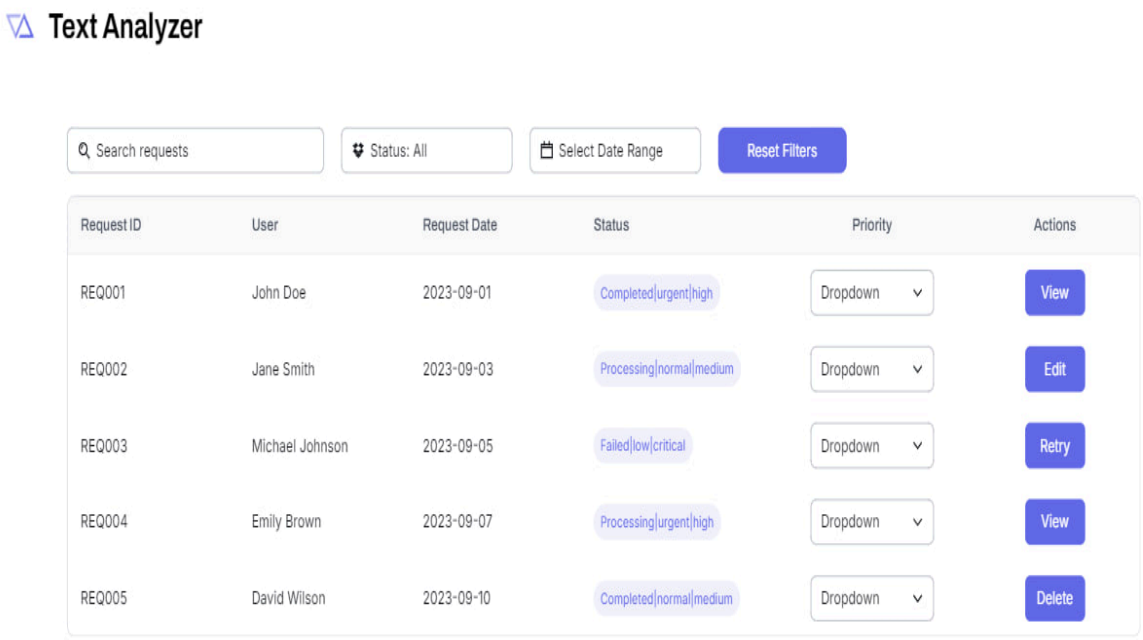

Рисунок 3.6 – Дизайн сторінки створення запиту на аналіз

Сторінка перегляду звіту аналізу (рис 3.7) є ключовою для виведення результатів роботи системи. Вона надає зручний спосіб перегляду результатів аналізу, включаючи тематичне моделювання та оцінки настроїв. Інтерфейс сторінки побудований так, щоб результати були представлені у вигляді зручних графіків, таблиць та текстових описів. Це допомагає користувачам легко інтерпретувати результати та приймати обґрунтовані рішення на основі отриманих даних.

Рисунок 3.7 – Дизайн сторінки перегляду звіту

Загалом, кожна сторінка була ретельно спроєктована з метою забезпечення інтуїтивно зрозумілого і візуально привабливого інтерфейсу, що відповідає потребам кінцевих користувачів. Макети цих сторінок закладають основу для розробки остаточного інтерфейсу системи, який забезпечить ефективність та зручність роботи з програмним забезпеченням.

3.3 Огляд стеку технологій

При розробці системи аналізу настроїв тексту на основні тематичного моделювання використано популярний MSS (Microsoft Simp Stack) стек технологій, до якого входять:

– мови програмування: C#, Typescript, Python;
– фреймворки: ASP.NET Core, Angular;
– СКБД: PostgresSQL.

Відповідний стек допомагає забезпечити стабільну роботу програми, масштабованість, безпеку даних та швидкість обробки запитів, що є особливо

важливим для систем, які займаються обробкою великих обсягів текстових даних та мають працювати в режимі реального часу. Правильний вибір технологій також впливає на інтеграцію з іншими сервісами та можливість розширення функціоналу в майбутньому.

3.3.1 Мови програмування

Вибір мови програмування C# (рис. 3.8) є доцільним для розробки бекенд-частини системи, оскільки вона надає широкі можливості для створення масштабованих вебзастосунків та мікросервісів завдяки платформі .NET. C# добре інтегрується з різними базами даних і сервісами, має високу продуктивність та підтримує асинхронне програмування, що важливо для обробки великих обсягів запитів та даних [21]. Також ця мова є популярною серед розробників, що полегшує залучення нових спеціалістів до проєкту та забезпечує доступ до великої кількості бібліотек і фреймворків.

Рисунок 3.8 – Логотип мови C#

TypeScript (рис. 3.9) використовується для розробки фронтенд-частини системи. Ця мова є розширенням JavaScript, яке додає статичну типізацію, що робить код більш структурованим та легшим у підтримці [21]. Завдяки своїй сумісності з JavaScript, TypeScript дозволяє використовувати широкий спектр

бібліотек і фреймворків, таких як Angular, що забезпечує зручну розробку інтерфейсу користувача. Статична типізація також допомагає зменшити кількість помилок під час розробки, що є критично важливим для створення складних взаємодій на фронтенді.

Рисунок 3.9 – Логотип мови Typescript

Python (рис. 3.10) є оптимальним вибором для реалізації алгоритмів машинного навчання, що використовуються в системі, зокрема для аналізу настроїв та тематичного моделювання тексту. Ця мова пропонує багатий набір бібліотек для роботи з даними, таких як scikit-learn, TensorFlow та pandas, що значно спрощує процес розробки та тестування моделей. Python також має простий синтаксис, що сприяє швидкому створенню прототипів і аналізу даних, дозволяючи скоротити час на розробку складних обчислювальних алгоритмів [?]. Це робить його незамінним інструментом для роботи з великими обсягами текстової інформації.

Рисунок 3.10 – Логотип мови Python

Обрані мови програмування – C#, TypeScript та Python – створюють збалансований та ефективний технологічний стек для розробки сучасної системи аналізу тексту на основі тематичного моделювання. C# забезпечує високу продуктивність та надійність на бекенді, що дозволяє ефективно обробляти запити та керувати бізнес-логікою системи. TypeScript сприяє створенню зручного та стабільного інтерфейсу користувача на фронтенді, зменшуючи ймовірність помилок завдяки статичній типізації. Python, у свою чергу, ідеально підходить для реалізації алгоритмів машинного навчання, забезпечуючи зручність роботи з великими обсягами даних і високий рівень підтримки для різних бібліотек. Такий підхід гарантує не лише високу продуктивність та масштабованість системи, а й забезпечує її гнучкість та легкість у подальшому розвитку й підтримці.

3.3.2 Front-end технології

Angular (рис. 3.11) є одним із найпопулярніших фронтенд-фреймворків для розробки вебзастосунків, створений і підтримуваний Google. Він базується на TypeScript, що забезпечує статичну типізацію та підвищену надійність коду. Angular використовує компонентний підхід, який дозволяє створювати модульні та повторно використовувані частини інтерфейсу, спрощуючи розробку та підтримку

великих застосунків [22]. Завдяки вбудованим можливостям для роботи з формами, маршрутизацією та HTTP-запитами, Angular забезпечує все необхідне для побудови динамічних та інтерактивних вебзастосунків.

Angular Material є бібліотекою компонентів, розробленою спеціально для Angular, що базується на принципах Material Design [23]. Ця бібліотека надає широкий набір готових компонентів, таких як кнопки, форми, модальні вікна, навігаційні панелі та інші елементи, які відповідають сучасним стандартам дизайну. Angular Material забезпечує зручний та консистентний стиль для інтерфейсу користувача, спрощуючи процес створення естетично привабливих і зручних інтерфейсів. Завдяки інтеграції з Angular, бібліотека легко налаштовується та розширюється, що дозволяє адаптувати її під специфічні вимоги проєкту.

Рисунок 3.11 – Логотипи Angular і Angular Material

Вибір Angular разом з Angular Material є доцільним завдяки їхній сумісності та зручності у використанні. Angular забезпечує стабільну основу для розробки складних вебзастосунків, дозволяючи створювати модульні та масштабовані системи, які легко підтримувати та розвивати. Використання Angular Material дозволяє значно скоротити час на розробку інтерфейсу завдяки готовим компонентам, які відповідають стандартам дизайну, що робить застосунок зручним і привабливим для користувачів. Такий підхід не лише прискорює процес

розробки, але й гарантує високу якість кінцевого продукту, забезпечуючи позитивний досвід взаємодії з користувачем.

3.3.3 Back-end технології

ASP.NET Core (рис. 3.12) є сучасним фреймворком для розробки вебзастосунків і API, створеним компанією Microsoft. Він є кросплатформним, що дозволяє запускати вебзастосунки на різних операційних системах, таких як Windows, Linux та macOS. ASP.NET Core відомий своєю високою продуктивністю та гнучкістю, що робить його ідеальним вибором для побудови масштабованих вебсервісів та API [24]. Він підтримує використання RESTful API, що забезпечує легке інтегрування з іншими компонентами системи, а також має вбудовану підтримку для авторизації, аутентифікації та захисту даних, що є важливим для безпеки застосунків.

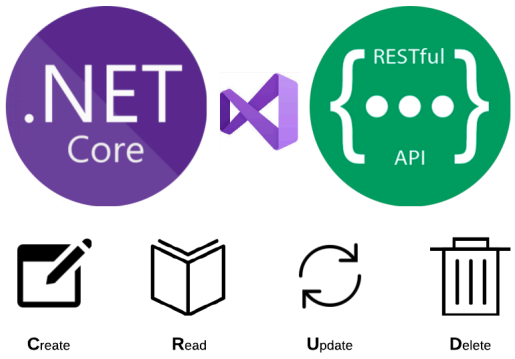

Рисунок 3.12 – Екосистема фреймворку ASP.NET Core

FastAPI – це фреймворк для створення вебзастосунків і API на базі Python, відомий своєю швидкістю та зручністю [25]. Він використовує сучасні можливості Python, такі як типізація, що допомагає розробникам швидко та ефективно писати код з високою продуктивністю. FastAPI (рис. 3.13) забезпечує автоматичну генерацію документації для API, що спрощує процес розробки та взаємодію з

іншими розробниками. Це робить його ідеальним для створення сервісів машинного навчання, де важливо швидко створювати прототипи, інтегрувати моделі та забезпечувати високу продуктивність на етапі впровадження.

Рисунок 3.13 – Логотип FastAPI

Python є основною мовою для роботи з машинним навчанням, а використання бібліотек sklearn і pandas значно полегшує реалізацію моделей та обробку даних. Scikit-learn (sklearn) забезпечує широкий набір алгоритмів для класифікації, регресії, кластеризації та інших завдань машинного навчання, що робить його ідеальним для виконання аналізу настроїв та тематичного моделювання. Pandas, у свою чергу, дозволяє ефективно працювати з великими наборами даних, виконувати їх очистку, трансформацію та аналіз, що є критичним на етапі підготовки даних для моделювання.

PostgreSQL, часто відома як Postgres (рис. 3.14), є потужною об’єктно-реляційною системою управління базами даних з відкритим кодом. Вона відрізняється своєю надійністю, масштабованістю та відповідністю стандартам SQL, що робить її одним з найбільш популярних виборів для зберігання та керування даними у складних застосунках. PostgreSQL підтримує як реляційні таблиці, так і роботу з JSON, що дозволяє зберігати структуровані та неструктуровані дані в одній системі [22]. Це робить її особливо привабливою для застосунків, де дані можуть бути представлені в різних форматах.

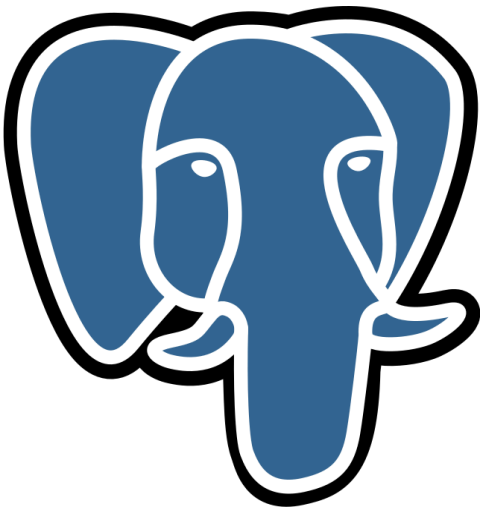

Рисунок 3.14 – Логотип СКБД PostgreSQL

Обраний технологічний стек для розробки системи включає C#, TypeScript, Python, фреймворки ASP.NET Core та Angular, а також базу даних PostgreSQL. Цей вибір є доцільним з кількох важливих причин. C# та ASP.NET Core забезпечують високу продуктивність та стабільність для розробки серверної частини, дозволяючи створювати надійні API, що відповідають вимогам швидкодії та масштабованості. TypeScript та Angular, у поєднанні з Angular Material, дозволяють створювати зручний і адаптивний інтерфейс користувача, який може легко обслуговувати різні пристрої, забезпечуючи при цьому сучасний і інтуїтивно зрозумілий користувацький досвід.

Python, разом із FastAPI, sklearn та pandas, виступає ключовим компонентом для реалізації машинного навчання та аналізу тексту, що є основою системи. Python відзначається великою кількістю бібліотек для обробки природної мови та машинного навчання, що дозволяє розробляти складні алгоритми аналізу настроїв та тематичного моделювання з мінімальними витратами часу та зусиль. FastAPI забезпечує високу продуктивність для інтеграції моделей машинного навчання у вигляді мікросервісів, що сприяє легкому розширенню можливостей системи.

PostgreSQL, як обрана база даних, дозволяє зберігати різні типи даних, що виникають в процесі роботи системи, включаючи результати аналізу та інформацію про користувачів. Її підтримка складних запитів, масштабованість та здатність працювати з великими обсягами даних роблять її ідеальним рішенням для забезпечення надійного та продуктивного зберігання. Загалом, поєднання цих технологій створює збалансовану та гнучку систему, яка відповідає вимогам до продуктивності, зручності користування та надійності, забезпечуючи ефективне виконання всіх функцій системи для аналізу настроїв тексту на основі тематичного моделювання.

3.3.4 Сторонні сервіси

Keycloak (рис. 3.15) – це відкрите рішення для управління ідентифікацією та доступом, яке призначене для обробки аутентифікації та авторизації користувачів. Воно надає можливість реалізації єдиного входу (SSO), що дозволяє користувачам проходити автентифікацію один раз і отримувати доступ до різних застосунків без повторного введення логіну та паролю [26]. Keycloak підтримує інтеграцію з різними постачальниками ідентифікації, LDAP-серверами та базами даних, а також забезпечує соціальний логін, дозволяючи користувачам входити через свої облікові записи в соціальних мережах.

Рисунок 3.15 – Логотип Keycloak

Важливою перевагою Keycloak є підтримка сучасних стандартів безпеки, таких як OAuth2.0, OpenID Connect та SAML, що забезпечує надійність та захищеність процесу аутентифікації. Крім того, Keycloak надає інструменти для управління користувачами, ролями та дозволами через адміністративну консоль, що дозволяє зручно налаштовувати права доступу до різних компонентів системи. Це особливо важливо в розроблюваних проєктах, які потребують детального контролю доступу та захисту даних користувачів.

Використання Keycloak у розробці системи аналізу настроїв тексту на основі тематичного моделювання обґрунтоване потребою в надійному управлінні доступом до застосунку та захисті конфіденційних даних. Keycloak значно спрощує процес інтеграції з іншими сервісами, оскільки усуває необхідність створення кастомних рішень для аутентифікації та авторизації, що економить час та ресурси. Він дозволяє налаштувати гнучкі сценарії доступу та взаємодії між користувачами та системою, зберігаючи високий рівень безпеки.

Висновки до розділу 3

У розділі 3 проведено детальний аналіз і розробку ключових аспектів системи, що є важливими для її успішного функціонування. Спочатку створено UML-діаграми класів та компонентів, що дозволило структурувати логіку роботи системи, визначити взаємозв’язки між основними класами та компонентами, а також краще зрозуміти внутрішню архітектуру проєкту. Ці діаграми допомагають чітко уявити структуру даних і забезпечують базу для подальшої реалізації бізнес-логіки.

Далі відбувся процес проєктування інтерфейсу користувача, де розроблені макети основних сторінок застосунку. Це дозволило створити зручний та інтуїтивно зрозумілий користувацький досвід, що сприятиме швидкому та ефективному доступу до функцій системи. Приділення уваги UX/UI забезпечує

легкість у використанні системи для користувачів, роблячи процес взаємодії з нею приємним та ефективним.

Крім того, у цьому розділі детально розглянуто вибір технологічного стеку для реалізації системи, включаючи C#, TypeScript, Python, а також фреймворки ASP.NET Core та Angular. Обраний стек технологій забезпечує оптимальний баланс між продуктивністю, масштабованістю та простотою розробки. Використання сучасних інструментів та бібліотек спрощує процес розробки, забезпечуючи можливість розширення функціональності та підтримки системи в майбутньому.

Таким чином, у розділі 3 закладено міцний фундамент для подальшої реалізації та розгортання системи. Створення UML-діаграм, ретельне проєктування інтерфейсу та вибір відповідного технологічного стеку дозволяють забезпечити структурований підхід до розробки, що сприятиме ефективній реалізації поставлених завдань та задоволенню потреб користувачів.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

4.1 Кодування програмних компонентів back-end частини

4.1.1 Налаштування та інгерація Keycloak з ASP.NET Core

Для налаштування та інтеграції Keycloak потрібно виконати кілька важливих кроків, кожен із яких має своє значення в забезпеченні безпеки та правильності взаємодії між компонентами системи.

Перший етап – це підняття Keycloak через Docker. Для цього створено Docker-контейнер, що містить всі необхідні компоненти для роботи Keycloak за допомогою команди, вказаної в офіційній документації (рис. 4.1).

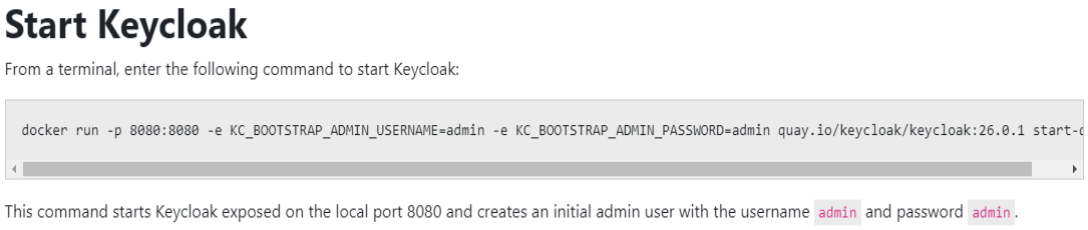

Рисунок 4.1 – Логотип Keycloak

Завдяки Docker, налаштування середовища є швидким і ефективним, оскільки всі залежності й конфігурації вже інтегровані у контейнер. Основною перевагою використання Docker є можливість запуску Keycloak без необхідності встановлення додаткових компонентів на локальну машину. Команда, що використовується для запуску контейнера, включала налаштування порту та вказівку на зберігання даних у внутрішньому сховищі для забезпечення постійності конфігурацій.

Наступний крок полягав у створенні нового realm (рис. 4.2). У Keycloak, realm є ізольованим простором, у межах якого зберігаються користувачі, ролі, клієнти й налаштування автентифікації. Кожен realm діє як окремий домен, що дозволяє розділити безпеку різних застосунків або середовищ. Створення realm

для проєкту дає можливість контролювати автентифікацію й авторизацію користувачів, які працюють з конкретним застосунком, ізольовано від інших систем. Під час створення realm налаштовано базові параметри, такі як ім’я, час життя сесій, методи автентифікації.

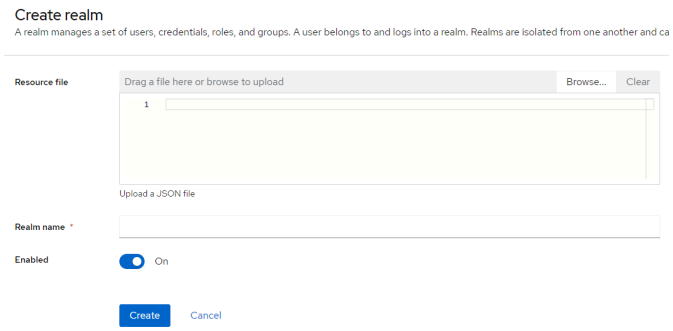

Рисунок 4.2 – Екран створення нового Realm

Наступним кроком є для фронтенду. Клієнти у Keycloak це програми або сервіси, які взаємодіють з Keycloak для автентифікації користувачів. Для фронтенду створено окремий клієнт (рис. 4.3), який використовує протокол OpenID Connect (OIDC) для управління авторизацією. Важливий етап налаштування – задання правильного редіректу після успішної авторизації користувача на фронтенд-застосунок.

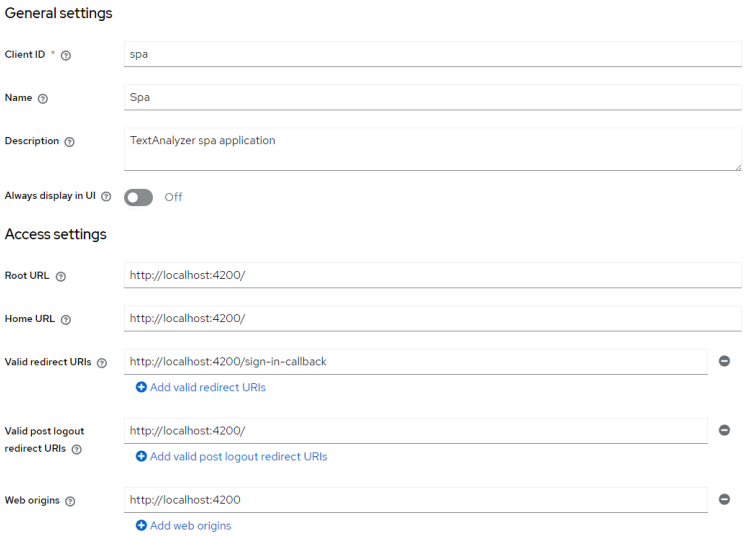

Рисунок 4.3 – Параметри конфігурації фронтенд-клієнта

Для бекенд-клієнта створено окремий сервісний клієнт, який взаємодіє з Keycloak через API (рис. 4.4). За допомогою цього клієнта бекенд може створювати нових та отримувати існуючих користувачів, і таким чином реалізований функціонал реєстрації користувача. Конфігурація включала налаштування клієнтських облікових даних та встановлення типу доступу через Client Credentials Flow для автоматизації взаємодії між сервісами.

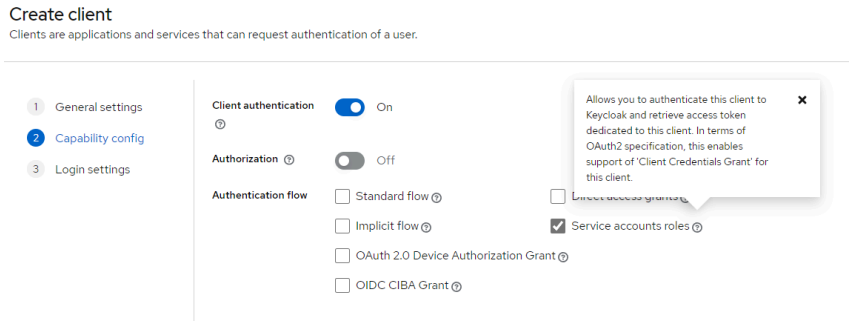

Рисунок 4.4 – Конфігурація бекенд-клієнта

На стороні бекенду для інтеграції авторизації за допомогою Keycloak використовувався пакет Microsoft.AspNetCore.Authentication.JwtBearer (рис. 4.5). Спочатку налаштовано механізм автентифікації, що використовує JWT-токени, видані Keycloak, для перевірки користувачів. У конфігурації ASP.NET Core додавалася підтримка аутентифікації, де вказувався URL Keycloak для отримання метаданих OpenID Connect, а також ідентифікатор клієнта, з яким сервер взаємодіє.

```
builder.Services // IServiceCollection
    .AddAuthentication()
    .AddJwtBearer(x :JwtBearerOptions ⇒
    {
        x.RequireHttpsMetadata = false;
        x.MetadataAddress = "http://localhost:8080/realms/TextAnalyzer/.well-known/openid-configuration";
        x.TokenValidationParameters = new TokenValidationParameters
        {
            ValidAudience = "api"
        };
    });
builder.Services.AddAuthorization();
```

Рисунок 4.5 – Інгерація валідації JWT токенів з Keycloak

Для створення клієнта для взаємодії з користувачами в реалмі Keycloak використовувалася бібліотека Refit (рис. 4.6), яка забезпечує простий і ефективний спосіб створення REST API-клієнтів. Спочатку був визначений інтерфейс для клієнта Keycloak, в якому описувалися всі необхідні ендпоінти для взаємодії з Keycloak API (рис. 4.7). В інтерфейсі зазначалися методи для отримання токенів, створення, оновлення та видалення користувачів, а також для отримання інформації про них.

Рисунок 4.6 – Логотип бібліотеки Refit

Далі за допомогою Refit клієнт генерувався автоматично, що дозволяло уникнути вручну написаного HTTP-коду. У конфігурації налаштовувалася базова URL-адреса Keycloak сервера, а також вказувалися необхідні заголовки для аутентифікації (Bearer-токен), отриманого з Keycloak. Для взаємодії з API був налаштований спеціальний обробник для автоматичного отримання нового токена доступу (рис. 4.8), якщо поточний токен ставав недійсним.

```
6 usages  1 inheritor
internal  interface IKeycloakUserApi
{
    [Get( path: "/admin/realms/{realm}/users")]
    1 usage  1 implementation
    Task<KeycloakUser[]> GetUsers(string realm, [Query] GetUsersFilters filters);

    [Post( path: "/admin/realms/{realm}/users")]
    1 usage  1 implementation
    Task CreateUser(string realm, [Body] CreateUserRequest request);
}
```

Рисунок 4.7 – Опис ендпоїнтів для взаємодії з користувачами

Клієнт дозволяв виконувати всі необхідні операції з користувачами через стандартні REST запити, такі як створення нових облікових записів, зміна атрибутів користувачів або їх видалення з системи. Refit спрощував взаємодію з API, перетворюючи HTTP-запити на звичайні методи C#, що значно полегшило складність інтеграції з Keycloak і зробило код чистішим і зрозумілішим.

```
internal async ValueTask<string> GetToken()
{
    var cachedToken = _cache.Get<string>(TokenCacheKey);
    if (cachedToken != null)
    {
        return cachedToken;
    }

    try
    {
        await _lock.WaitAsync();

        cachedToken = _cache.Get<string>(TokenCacheKey);
        if (cachedToken != null)
        {
            return cachedToken;
        }

        var request = new OidcTokenRequest
        {
            ClientId = _options.ClientId,
            ClientSecret = _options.ClientSecret
        };
        var response = await _keycloakOidcApi.GetToken(_options.Realm, request);

        _cache.Set(
            TokenCacheKey,
            response.AccessToken,
            absoluteExpiration: _timeProvider.GetUtcNow().AddSeconds(response.ExpiresIn)
        );

        return response.AccessToken;
    }
    finally
    {
        _lock.Release();
    }
```

Рисунок 4.8 – Логіка отримання та оновлення токену доступу

Завдяки інтеграції Keycloak забезпечено захист API через JWT токени, а Refit дозволив легко реалізувати клієнт для управління користувачами Keycloak. Інтеграція значно спростила управління користувачами та забезпечує високий рівень безпеки для системи, відповідно до вимог.

4.1.2 Запит на аналіз тексту

Запит на аналіз тексту є ключовим елементом системи, оскільки він забезпечує основну взаємодію користувача з аналітичними можливостями платформи. Завдяки цьому функціоналу користувач може надати тексти для аналізу, що запускає процес обробки даних, включаючи аналіз настроїв і тематичне моделювання.

У системі, даний запит доступний, як REST-ендпоїнт, що має наступний вигляд (рис. 4.9):

```
public static IEndpointRouteBuilder MapAnalysisEndpoints(this IEndpointRouteBuilder endpoints)
{
    var analGroup = endpoints.MapGroup(prefix: "/analysis");

    analGroup.MapGet(pattern: "/", GetTextAnalysisRequests);
    analGroup.MapGet(pattern: "/{id:guid}", GetTextAnalysisRequest);
    analGroup.MapPost(pattern: "/", RequestTextAnalysis);

    return endpoints;
}

1 usage
private static async Task<Ok> RequestTextAnalysis(
    [FromBody] CreateTextAnalysisRequest request,
    [FromServices] IMediator mediator,
    [FromServices] IMapper mapper)
{
    var command = mapper.Map<RequestTextAnalysisCommand>(request);
    await mediator.Send(command);
    return TypedResults.Ok();
}
```

Рисунок 4.9 – Опис ендпоїнту для аналізу тексту

Окремий метод RequestTextAnalysis відповідає за обробку POST-запитів для створення нового запиту на аналіз тексту. Він приймає вхідні дані у вигляді об’єкта CreateTextAnalysisRequest, що надходить через HTTP-запит. У коді використовується бібліотека IMediator, щоб відправити команду RequestTextAnalysisCommand, яка запускає необхідну бізнес-логіку для обробки

запиту на аналіз тексту. Після цього метод повертає результат з відповіддю Ok, що свідчить про успішне виконання операції.

Важливо відзначити, що ендпоїнт RequestTextAnalysis не займається безпосередньо аналізом тексту. Його основна функція полягає в тому, щоб прийняти запит від користувача, додати відповідний запис у базу даних і ініціювати подальший процес шляхом відправки повідомлення до брокера повідомлень (рис. 4.10).

```
public async Task Handle(RequestTextAnalysisCommand request, CancellationToken cancellationToken)
{
    var analysisRequest = new AnalysisRequest
    {
        Corpus = request.Corpus,
        RequestedAt = _timeProvider.GetUtcNow(),
        UserId = _currentUser.Id
    };
    _dbContext.AnalysisRequests.Add(analysisRequest);
    await _dbContext.SaveChangesAsync();

    var @event = new TextAnalysisRequested { AnalysisId = analysisRequest.Id };
    await _bus.Publish(@event);
}
```

Рисунок 4.10 – Обробка команди на запит аналізу тексту

При обробці запиту на аналіз тексту в базі даних створюється запис із необхідною інформацією, зокрема, дані про користувача, що ініціював запит, і сам текст, що підлягає аналізу. Після цього ендпоїнт відправляє повідомлення до брокера повідомлень, яке повідомляє інші компоненти системи, що новий запит на аналіз очікує обробки.

Таким чином, основне завдання цього ендпоїнта – забезпечити асинхронну обробку. Це дозволяє системі ефективно обробляти великі обсяги запитів і не блокувати користувацькі дії під час виконання складних обчислень, що підвищує загальну продуктивність і стабільність роботи системи.

У класі AnalyzeTextCommandHandler реалізується обробка команди аналізу тексту, яка викликається для виконання процесу аналізу. У конструкторі класу відбувається впровадження необхідних сервісів, таких як ITextAnalysisService, AppDbContext та IMapper. Ці сервіси забезпечують доступ до бази даних, виконання аналітичних операцій та перетворення об’єктів, що відіграє ключову роль у подальшій обробці даних. Метод Handle виконує основну логіку аналізу (рис. 4.11).

```
public async Task<OneOf<Success, Error>> Handle(AnalyzeTextCommand request, CancellationToken cancellationToken)
{
    var analysis = await _dbContext.AnalysisRequests.FirstOrDefaultAsync(
        x:AnalysisRequest => x.Id == request.AnalysisRequestId
    ); // Task<AnalysisRequest?>
    if (analysis?.Status != AnalysisRequestStatus.New){...}

    var payload = new TextPayload
    {...};

    var sentimentAnalysisTask = _analysisService.SentimentAnalysis(payload);
    var topicModelingTask = _analysisService.TopicModeling(payload);

    await Task.WhenAll(sentimentAnalysisTask, topicModelingTask);

    var sentimentAnalysis:OneOf<SentimentAnalysisResult,...> = await sentimentAnalysisTask;
    if (!sentimentAnalysis.TryPickT0(out var sentimentResult, out var sentimentError)){...}

    analysis.AnalysisResults.Add(item: _mapper.Map<SentimentAnalysisResult>(sentimentResult));

    var topicModeling:OneOf<TopicModelingResult,...> = await topicModelingTask;
    if (!topicModeling.TryPickT0(out var topicModelingResult, out var topicModelingError)){...}

    analysis.AnalysisResults.Add(item: _mapper.Map<TopicModelingAnalysisResult>(topicModelingResult));

    analysis.Status = AnalysisRequestStatus.Completed;
    await _dbContext.SaveChangesAsync();

    return new Success();
}
```

Рисунок 4.11 – Метод для обробки аналізу тексту

Спочатку відбувається отримання запиту на аналіз тексту з бази даних за ідентифікатором AnalysisRequestId. Якщо статус запиту не є новим, обробка завершується, і повертається успіх. У разі, якщо статус є новим, формується об’єкт TextPayload, що містить текст для аналізу. Після цього запускаються дві

асинхронні задачі для проведення аналізу настроїв і тематичного моделювання, які виконуються паралельно за допомогою Task.WhenAll.

Коли обидві задачі завершуються, результати перевіряються. Якщо виникає помилка під час одного з аналізів, статус запиту оновлюється на «Faulted», дані зберігаються в базі, і повертається помилка. Якщо все пройшло успішно, результати аналізів зберігаються в базі даних, і статус запиту оновлюється на «Completed». Таким чином, цей клас відповідає за обробку аналітичних запитів і управління їх станом.

Така реалізація функціоналу запиту на аналіз тексту має безліч переваг, зокрема абстракція від методів аналізу через інтерфейс ITextAnalysisService, а також розвантаження інтерфейсу користувача, який не повинен чекати на обробку запиту, а може отримати результат пізніше за допомогою техніки Long Polling.

4.2 Тренування моделей машинного навчання

Метод опорних векторів (SVC) обрано для аналізу настроїв завдяки його високій точності при роботі з різними наборами даних та здатності ефективно класифікувати текстові дані. SVC добре підходить для задач з невеликою кількістю класів, оскільки цей алгоритм створює оптимальні гіперплощини, які розділяють класи, що дозволяє досягти кращих результатів навіть при невеликій кількості тренувальних даних [27]. Також важливою перевагою SVC є його стійкість до шуму у даних.

У методі опорних векторів основна ідея полягає у пошуку гіперплощини, яка розділяє два класи так, щоб максимізувати щілину між ними. Для лінійно роздільних даних, шукається гіперплощина, яка має рівняння, де w – вектор ваг, x – вектор ознак, а b – зсув:

$$w^T x + b = 0.$$

Алгоритм шукає таку гіперплощину, яка забезпечує максимальну щілину між точками обох класів. Це досягається шляхом мінімізації величини вектора ваг w, що еквівалентно максимізації зазору між класами. Граничні умови для кожної точки x_i із класу y_i (де $y_i = \pm 1$) визначаються як:

$$y_i(w^T x_i + b) \geq 1.$$

Це означає, що всі точки одного класу повинні знаходитися на одній стороні гіперплощини, а точки іншого класу – на протилежній, причому найближчі точки повинні бути на відстані не менше ніж 1 від гіперплощини. Мета – мінімізувати функцію:

$$\frac{1}{2}||w||^2.$$

Однак, якщо дані не є лінійно роздільними, вводяться додаткові змінні для помилок ξ_i, щоб дозволити певним точкам бути класифікованими неправильно. Тоді умови модифікуються до:

$$y_i(w^T x_i + b) \geq 1 - \xi_i$$

А нова функція, яку треба мінізувати:

$$\frac{1}{2}||w||^2 + C\sum_i \xi_i,$$

де C – гіперпараметр, який контролює компроміс між максимізацією щілини та кількістю неправильно класифікованих точок.

Під час тренування моделі машинного навчання, використано реалізацію методу SVC з бібліотеки sklearn, а також допоміжні бібліотеки, як pandas та nltk.

Датасет завантажувався з CSV-файлу, а для спрощення з нього відбираються лише два стовпці: текст і мітки. Мітки були перетворені з числових у текстові мітки, а потім вони закодовані в числовий формат за допомогою LabelEncoder, що потрібно для тренування моделі.

```
df = pd.read_csv('./data/data.csv', names=['label', 'id', 'date', 'flag', 'user', 'text'])
df = df[['text', 'label']]
df['label'] = df['label'].apply(lambda x: 'negative' if x == 0 else 'positive')
label_encoder = LabelEncoder()
df['label'] = label_encoder.fit_transform(df['label'])
```

Для отримання точніших результатів важливо зробити попередню обробку даних, для цього завантажуються англійські стоп-слова з бібліотеки nltk, а для стемінгу використовується SnowballStemmer. Також створена функція для передобробки текстових рядків:

```
nltk.download('stopwords')

stemmer = SnowballStemmer('english')
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'[^a-z\s]', '', text)
    tokens = text.split()
    tokens = [stemmer.stem(token) for token in tokens if token not in stop_words]
    return ' '.join(tokens)
df['text'] = df['text'].apply(preprocess_text)
```

Дані були поділені на тренувальні та тестові з допомогою train_test_split, що дозволяє уникнути зайвого тренування моделі, отримувати стабільні результати під час тестів та перевірити її на окремому наборі. Для побудови моделі використано Pipeline, що поєднує два етапи: векторизація тексту з допомогою TfidfVectorizer і тренування моделі SVC з лінійним ядром, а після тренування моделі здійснюється прогнозування для тестової вибірки, а потім розраховується точність на основі прогнозованих і реальних значень.

```
X_train, X_test, y_train, y_test = train_test_split(
                        pd.concat([df['text'][:10000], df['text'][-10000:]]),
pd.concat([df['label'][:10000], df['label'][-10000:]]), test_size=0.2,
random_state=21910825
)
```

```
pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(max_features=5000, ngram_range=(1, 2))),
    ('svm', SVC(kernel='linear', probability=True))
])

pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Model Accuracy: {accuracy * 100:.2f}%')
```

Метод латентного розміщення Діріхле (LDA) обраний для тематичного моделювання через його здатність автоматично знаходити приховані теми в текстах, що робить його ідеальним для роботи з текстовими корпусами різного розміру [28]. LDA використовує ймовірнісну модель, яка визначає набір тем, що найкраще описують текстовий корпус, дозволяючи виявляти важливі теми та їх ключові слова. Завдяки цій моделі система може проводити глибокий аналіз текстових даних, що корисно для вивчення структурованих патернів у великих масивах інформації.

Основна ідея LDA полягає в тому, що документи можна уявити як суміші прихованих тем, а кожна тема представлена набором слів з певними ймовірностями. LDA шукає ці приховані теми, на основі яких можна зрозуміти структуру тексту.

Математично, для кожного документа d і теми k LDA моделює ймовірність того, що певне слово належить до цієї теми. При цьому документ представлений як суміш тем, а кожна тема – суміш слів.

Процес включає наступні етапи:

– для кожного документа d обирається розподіл тем θ_d, який є параметром Діріхле;

– для кожного слова у документі обирається тема z, виходячи з розподілу тем θ_d;

– обирається слово для теми z, виходячи з іншого розподілу Діріхле ϕ_k, який моделює ймовірність слів для кожної теми.

Ключовою формулою в LDA є розрахунок ймовірності слова в документі через суму по всіх можливих темах:

$$P(W_i|d) = \sum_{k=1}^{K} P(w_i|z_k)P(z_k|d)$$

У реалізації тренування моделі LDA основні етапи включають завантаження даних, їх попередню обробку, векторизацію, а також побудову моделі LDA для виявлення тем. На початку відбувається імпорт необхідних бібліотек, серед яких nltk для обробки тексту та sklearn, яка надає імплементацію моделі LDA. Дані, як і у випадку з аналізом настроїв читаються з CSV-файлу, і для спрощення береться лише стовпець з текстом.

```
nltk.download('stopwords')

stemmer = PorterStemmer()
stop_words = set(stopwords.words('english'))
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'[^a-z\s]', '', text)
    tokens = text.split()
    tokens = [stemmer.stem(token) for token in tokens if token not in stop_words]
    return ' '.join(tokens)
df = pd.read_csv('./data/data.csv', names=['label', 'id', 'date', 'flag', 'user',
'text'])
df = df[['text']]
df['text'] = df['text'].apply(preprocess_text)
```

Коли дані оброблені, вони розділяються на тренувальний і тестовий набори за допомогою train_test_split та створюється Pipeline, що об'єднує векторизацію тексту з допомогою TfidfVectorizer і побудову моделі LDA через LatentDirichletAllocation. Вибір параметра n_components у якості 3 означає, що модель намагається знайти три основні теми у тексті.

```
train, test = train_test_split(
             pd.concat([df['text'][:10000],   df['text'][-10000:]]),   test_size=0.2,
random_state=21910825
)
pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(max_features=5000, ngram_range=(1, 2))),
    ('lda', LatentDirichletAllocation(n_components=3, random_state=21910825))
])
pipeline.fit(train)
feature_names = pipeline.named_steps['vectorizer'].get_feature_names_out()
model = pipeline.named_steps['lda']
for topic_idx, topic in enumerate(model.components_):
        top_features_ind = topic.argsort()[:-10 - 1:-1]
        top_features = [feature_names[i] for i in top_features_ind]
        print(f"Topic {topic_idx}: {top_features}")
```

Після навчання моделі виводяться ключові слова для кожної теми, що дозволяє зрозуміти основний зміст тем, які були виявлені LDA. Використовуючи model.components_, можна отримати індекси найбільш значущих слів для кожної теми, що допомагає в подальшому аналізі тексту. Таким чином, завершено навчання та аналіз тем, наданих в текстових даних, що є важливим етапом у розробці системи аналізу тексту.

Основне API системи реалізовано на C#, що забезпечує інтеграцію з усіма необхідними функціональними модулями. Однак, для використання моделей машинного навчання, розроблено окремий застосунок на базі FastAPI. Цей підхід дозволяє більш ефективно управляти специфічними задачами, пов'язаними з обробкою даних та запуском моделей машинного навчання.

Застосунок на FastAPI **(Додаток Б)** містить спеціально спроєктовані ендпоїнти, які дозволяють взаємодіяти з моделями, завантажуючи їх безпосередньо з диску. Ці ендпоїнти забезпечують функціональність для виконання аналізу тексту, зокрема, для запуску тренованих моделей SVC і LDA на вхідних даних. Це забезпечує гнучкість у роботі з моделями, а також спрощує інтеграцію нових алгоритмів і методів у майбутньому.

4.3 Кодування програмних компонентів front-end частин

4.3.1 Налаштування авторизації та автентифікації

В першу чергу, для коректної роботи застосунку, треба налаштувати авторизацію та автентифікацію, а для полегшення даної задачі існує бібліотека angular-oidc-client, яка може працювати з різними OIDC-провайдерами, в тому числі і з keycloak.

```
export const appConfig: ApplicationConfig = {
 providers: [
   provideZoneChangeDetection({ eventCoalescing: true }),
   provideRouter(appRoutes),
   provideHttpClient(
     withFetch()
   ),
   provideAuth({
     config: {
       authority: 'http://localhost:8080/realms/TextAnalyzer',
       redirectUrl: `${window.location.origin}/sign-in-callback`,
       postLogoutRedirectUri: window.location.origin,
       clientId: 'spa',
       scope: 'openid profile email',
       responseType: 'code',
       silentRenew: true,
       useRefreshToken: true,
       renewTimeBeforeTokenExpiresInSeconds: 30
     }
   }),
   provideAnimationsAsync()
 ]
};
```

Конфігурація appConfig у цьому коді визначає базові налаштування для авторизації та автентифікації на фронтенді за допомогою OIDC (OpenID Connect) протоколу. Через функцію provideAuth вказуються параметри для інтеграції з Keycloak. Параметр authority вказує адресу сервера автентифікації, де розташовано реалм TextAnalyzer.

Redirect url налаштовано на отримання відповідей після входу, а також є вказівка на клієнт з ідентифікатором spa, який використовує scopes openid, profile та email для доступу до певних даних користувача. ResponseType: ‘code’ налаштовує потік авторизації, а silentRenew забезпечує автоматичне оновлення токена у фоновому режимі за 30 секунд до його закінчення, зберігаючи сесію активною без переривань.

```
@Injectable({
 providedIn: 'root'
})
export class AuthService {
 private _oidcService = inject(OidcSecurityService);
 private _apiService = inject(ApiService);

 public init(): Observable<LoginResponse> {
   return this._oidcService.checkAuth();
 }

 public signin(): void {
   this._oidcService.authorize();
 }

 public signup(request: CreateUserRequest): Observable<void> {
   return this._apiService.createUser(request);
 }

 public signout(): Observable<unknown> {
   return this._oidcService.logoff();
 }
}
```

AuthService реалізує всі методи для взаємодії з авторизаційними функціями. Використання OidcSecurityService у даному випадку дозволяє працювати з автентифікацією та управлінням сесією користувача. Метод init() перевіряє поточний статус авторизації, зокрема наявність активної сесії, та повертає об’єкт з інформацією про логін. Метод signin() активує процес автентифікації, перенаправляючи користувача на сторінку входу, а signup() надсилає запит на

створення нового користувача через сервіс API. Метод signout() завершує сесію і повертає Observable для можливих подальших дій після виходу.

```
const redirectToDashboard = (router: Router) => {
 const loginPath = router.parseUrl('/dashboard');
 return new RedirectCommand(loginPath);
};

export const anonymousOnlyGuard: CanActivateFn = () => {
 const authService = inject(OidcSecurityService);
 const router = inject(Router);

 return authService.isAuthenticated$.pipe(
   map(x => x.isAuthenticated ? redirectToDashboard(router) : true)
 );
};
```

AnonymousOnlyGuard – перевіряє, чи вже виконано вхід до системи. Якщо користувач вже аутентифікований, виконується перенаправлення на /dashboard за допомогою функції redirectToDashboard. Ця функція визначає вхідну точку, де користувачеві стає доступним основний функціонал. Якщо ж користувач не авторизований, йому дозволяється продовжити шлях на сторінку для гостей або сторінку входу, забезпечуючи захист компонентів для неавторизованих відвідувачів.

```
export const appRoutes: Route[] = [
 { path: 'sign-in-callback', component: AuthCallbackComponent },
 { path: '', component: LandingComponent, canActivate: [anonymousOnlyGuard] },
 {
   path: 'dashboard',
   component: DashboardComponent,
   canActivate: [autoLoginPartialRoutesGuard],
   canActivateChild: [autoLoginPartialRoutesGuard],
   children: [
     { path: 'new-analysis', component: RequestAnalysisComponent },
     { path: 'analysis-requests', component: AnalysisRequestsComponent }
   ]
 },
 { path: '**', pathMatch: 'full', redirectTo: '' }
];
```

Конфігурація маршрутів містить усі необхідні шляхи програми, контролюючи доступ до сторінок залежно від статусу авторизації. При зверненні

до sign-in-callback користувач перенаправляється на компонент AuthCallbackComponent після успішної автентифікації.

Основна сторінка LandingComponent захищена Guard anonymousOnlyGuard, що дозволяє її відвідувати тільки неавторизованим користувачам.

Захист для dashboard реалізовано з використанням autoLoginPartialRoutesGuard, який автоматично перенаправляє на сторінку входу, якщо доступ намагається отримати неавторизований користувач. У dashboard розміщені дочірні маршрути для сторінок new-analysis та analysis-requests, доступ до яких можливий лише для авторизованих користувачів.

4.3.2 Сторінка створення запиту на аналіз тексту

Сторінка створення запиту на аналіз тексту реалізована за допомогою Angular Material, що забезпечує сучасний і зручний інтерфейс для користувачів [20]. Основна структура сторінки складається з контейнера, що містить картку (mat-card), в якій розміщені заголовок, форма та кнопки для взаємодії.

На початку картки знаходиться заголовок, який інформує користувача про те, що це сторінка для створення нового запиту на аналіз. Далі розташована форма, що прив’язана до FormGroup, яка забезпечує контроль над введеними даними. При відправленні форми викликається метод requestTextAnalysis(), який обробляє введені дані.

Форма включає mat-form-field, який містить текстове поле для введення тексту, який потрібно проаналізувати. Поле має підказку, що показує кількість символів, введених у полі, відносно максимального дозволеного значення, визначеного в константі TEXT_MAX_LENGTH. Це забезпечує зручність та інформативність для користувача.

Крім введення тексту, форма також пропонує альтернативний спосіб завантаження текстового документа через кнопку "Upload a Document". При

натисканні на цю кнопку відкривається діалогове вікно для вибору файлу, яке за допомогою Angular активується через приховане поле вводу типу file. Це дозволяє користувачу завантажити текстовий файл, вміст якого буде оброблено для аналізу.

На завершення, кнопка «Submit» дозволяє відправити запит на аналіз, сповіщаючи систему про намір користувача виконати аналіз введеного тексту. Завдяки використанню Angular Material, сторінка виглядає естетично привабливо та є зручною у використанні, що підвищує користувацький досвід.

```
<div class="container">
 <mat-card class="text-analyzer-card">
   <h1 class="title">New text analysis</h1>

   <form [formGroup]="analysisForm" (ngSubmit)="requestTextAnalysis()">
     <mat-form-field appearance="outline" class="input-field">
       <mat-label>Paste Text for Analysis</mat-label>
                              <textarea    matInput    rows="5"    placeholder="Input    text"
formControlName="corpus"></textarea>
         <mat-hint align="end">{{ analysisForm.value.corpus?.length }} / {{ TEXT_MAX_LENGTH
}}</mat-hint>
     </mat-form-field>

     <p><strong>---OR---</strong></p>

     <div class="file-upload">
       <input type="file" accept=".txt" (change)="getFileText($event)" hidden #fileInput>

       <button mat-button color="primary" class="upload-button" (click)="fileInput.click()">
         <mat-icon>upload_file</mat-icon>
         Upload a Document
       </button>
     </div>

     <button mat-raised-button type="submit" color="primary" class="submit-button">
       <mat-icon>send</mat-icon>
       Submit
     </button>
   </form>
 </mat-card>
</div>
```

Компонент RequestAnalysisComponent відповідає за реалізацію логіки сторінки для створення запиту на аналіз тексту. Вона використовує декоратор @Component, який визначає селектор, імпорти необхідних модулів, шаблон та стилі для компонента. Селектор app-request-analysis дозволяє використовувати компонент в інших частинах застосунку.

В компоненті визначена константа TEXT_MAX_LENGTH, яка задає максимальну довжину тексту для аналізу, що становить 4096 символів. Це обмеження допомагає уникнути перевантаження системи великою кількістю даних та забезпечує коректну роботу аналізу.

Клас компонента реалізує реактивну форму за допомогою NonNullableFormBuilder, який інжектується для створення форми analysisForm. Ця форма містить одне поле corpus, яке є обов’язковим для заповнення та має обмеження за довжиною, згідно з визначеною константою. Всі валідації поля виконуються автоматично, завдяки чому користувачеві не буде дозволено відправити форму з невірними даними.

Метод getFileText активується при виборі файлу користувачем. Він перевіряє, чи був обраний файл, а потім читає його вміст. Використовуючи await file.text(), метод асинхронно отримує текст з файлу. Після цього вміст файлу присвоюється полю corpus форми, що дозволяє користувачу легко завантажити текст для аналізу замість його ручного введення.

Коли користувач натискає кнопку "Submit", викликається метод requestTextAnalysis. Цей метод спочатку перевіряє, чи є форма валідною. Якщо форма невірна, відправлення запиту не відбувається. У разі валідності, дані з форми беруться за допомогою getRawValue(), і викликається метод сервісу _apiService.requestTextAnalysis(formValue), який виконує запит на сервер для обробки аналізу тексту. Після успішного виконання запиту користувача

перенаправляють на сторінку запитів аналізу в рамках інтерфейсу, використовуючи Router.

```
@Component({
 selector: 'app-request-analysis',
 standalone: true,
  imports: [CommonModule, MatIcon, MatButton, MatLabel, MatFormField, MatInput, MatCard,
ReactiveFormsModule, MatHint],
 templateUrl: './request-analysis.component.html',
 styleUrl: './request-analysis.component.scss'
})
export class RequestAnalysisComponent {
 public readonly TEXT_MAX_LENGTH = 4096;

 private _fb = inject(NonNullableFormBuilder);
 private _apiService = inject(ApiService);
 private _destroyRef = inject(DestroyRef);
 private _router = inject(Router);

 public analysisForm = this._fb.group({
                          corpus:         this._fb.control('',         [Validators.required,
Validators.maxLength(this.TEXT_MAX_LENGTH)])
 });

 public async getFileText(event: Event): Promise<void> {
   const input = event.target as HTMLInputElement;
   if (!input.files?.length) {
     return;
   }

   const file = input.files[0];
   const text = await file.text();
   console.log(text);

   this.analysisForm.controls.corpus.setValue(text);
 }

 public requestTextAnalysis(): void {
   if (this.analysisForm.invalid) {
     return;
   }

   const formValue = this.analysisForm.getRawValue();
```

```
    this._apiService
      .requestTextAnalysis(formValue)
      .pipe(takeUntilDestroyed(this._destroyRef))
      .subscribe(() => this._router.navigate(['/dashboard/analysis-requests']));
  }
}
```

Таким чином, цей компонент забезпечує повний цикл роботи з запитом на аналіз тексту, починаючи від введення даних і завантаження файлу до відправлення запиту на сервер і перенаправлення користувача на відповідну сторінку.

Висновки до розділу 4

У розділі 4 детально розглянуто всі ключові компоненти, які формують архітектуру системи. Перш за все акцент був зосереджений на кодуванні програмних компонентів бекенд-частини. Важливою частиною цього процесу стало налаштування та інтеграція Keycloak з ASP.NET Core, що забезпечило механізм авторизації та автентифікації користувачів, необхідний для безпеки системи. Клієнти для фронтенду та сервісний клієнт для бекенду були налаштовані з урахуванням специфіки проєкту, що спростило взаємодію між компонентами системи.

Також реалізовано ендпоїнт для запиту на аналіз тексту, який не тільки створює запис у базі даних, а й відправляє повідомлення до брокера для подальшої обробки. Основна логіка аналізу тексту виявилася реалізованою у класі AnalyzeTextCommandHandler, де обробка запиту включала виконання двох основних завдань: аналізу настроїв і тематичного моделювання. Цей підхід дозволив чітко структурувати код і відокремити відповідальності між компонентами.

Далі розглянуто процес тренування моделей машинного навчання з використанням SVC та LDA. Для SVC обрано метод, який демонструє високу

точність у задачах класифікації, тоді як LDA був використаний для тематичного моделювання, що допомогло виявити приховані теми в тексті. Ці моделі були реалізовані за допомогою Python, що забезпечило гнучкість і зручність для подальшої роботи з ними.

В останню чергу розглянуто кодування програмних компонентів фронтенду, де особлива увага приділена налаштуванню авторизації та автентифікації. Відповідно до специфікацій, компоненти реалізовані з використанням Angular, включаючи компоненти для створення запитів на аналіз тексту. Сторінка для створення запиту на аналіз тексту забезпечує користувачеві зручний інтерфейс для введення даних, підтримуючи функціонал завантаження файлів та введення тексту вручну.

У підсумку, розділ демонструє комплексний підхід до реалізації системи, з акцентом на інтеграцію бекенд і фронтенд-частин, а також на впровадження моделей машинного навчання, що формує потужний інструмент для аналізу тексту.

ВИСНОВКИ

В результаті виконання кваліфікаційної магістерської роботи систему аналізу настроїв тексту на основі тематичного моделювання, що забезпечує автоматизований процес аналізу текстів за допомогою сучасних методів машинного навчання. Основною мета проєкту це створення програмного забезпечення, яке не тільки виконує аналіз тексту, але й надає зручний інтерфейс для користувачів.

Для досягнення поставленої мети вирішено ряд завдань:

- проаналізовано предметну область і аналоги ПЗ;
- визначено функціонал системи;
- створено блок-схеми та діаграми роботи системи;
- зібрано та проаналізовано методичні рекомендації;
- розроблена frontend-частина системи на базі фреймворку Angular;
- розроблена backend-частина системи на базі фреймворку ASP.NET;
- розроблена ML-частина системи на базі бібліотеки sklearn.

Результатом проведеної роботи є система, здатна обробляти запити користувачів і надавати результати аналізу в зручному форматі. Проєкт демонструє, як сучасні технології машинного навчання можуть бути інтегровані в програмні рішення, покращуючи процеси аналізу тексту та забезпечуючи нові можливості для користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Practical Machine Learning with Python. URL: https://link.springer.com/book/10.1007/978-1-4842-3207-1 (дата звернення: 07.05.2024).

2. Bird S., Klein E., Loper E. Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit. O'Reilly Media, Inc., 2009. 506 p. ISBN 978-0-596-55571-9.

3. Prudhvi K., Bharath Chowdary A., Subba Rami Reddy P Text Summarization Using Natural Language Processing. Intelligent System Design(Singapore, 2021). Singapore : Springer, 2021. DOI:10.1007/978-981-15-5400-1_54. pp. 535–547.

4. Mertz D. Text Processing in Python. Addison-Wesley Professional, 2003. 544 p. ISBN 978-0-321-11254-5.

5. JUGRAN S., KUMAR A., TYAGI B. S. et al. Extractive Automatic Text Summarization using SpaCy in Python & NLP. 2021 International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)2021 International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE). (03.2021). DOI:10.1109/ICACITE51222.2021.9404712. pp. 582–585.

6. Thanaki J. Python Natural Language Processing. Packt Publishing Ltd, 2017. 476 p. ISBN 978-1-78728-552-1.

7. Vasiliev Y. Natural Language Processing with Python and spaCy: A Practical Introduction. No Starch Press, 2020. 217 p. ISBN 978-1-71850-053-2.

8. Kaur C., Sharma A. Social Issues Sentiment Analysis using Python. 2020 5th International Conference on Computing, Communication and Security (ICCCS)2020 5th International Conference on Computing, Communication and Security (ICCCS). (10.2020). DOI:10.1109/ICCCS49678.2020.9277251. P. 1–6.

9. Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, Luke Zettlemoyer. Deep Contextualized Word Representations. NAACL, 2018. 15 p.

10. Medallia Products. URL: https://www.medallia.com/products (дата звернення: 11.05.2024).

11. Lexalitics: Top Applications of Sentiment Analysis & Text Analytics URL: https://www.lexalytics.com/applications (дата звернення: 11.05.2024).

12. Qualtrics: Customer experiences. URL: https://www.qualtrics.com/customer-experience (дата звернення: 11.05.2024).

13. What are your ERD needs? URL: https://www.lucidchart.com/pages/er-diagrams (дата звернення 25.11.2024).

14. What is a Data Flow Diagram? URL: https://www.lucidchart.com/pages/data-flow-diagram (дата звернення: 25.11.2024).

15. The Complete Guide To Understand IDEF Diagram. URL: https://www.edrawmax.com/article/the-complete-guide-to-understand-idef-diagram.html (дата звернення 25.11.2024).

16. Coulouris G. F., Dollimore J., Kindberg T. Distributed Systems: Concepts and Design. Pearson Education, 2005. 952 p. ISBN 978-0-321-26354-4.

17. Distributed systems. ACM Other Books. URL: https://dl.acm.org/doi/abs/10.1145/90417 (дата звернення: 07.05.2024).

18. What is Deployment Diagram? URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-deployment-diagram/ (дата звернення: 25.11.2024).

19. UML Diagram Types Guide: Learn About All Types of UML Diagrams with Examples. URL: https://creately.com/blog/diagrams/uml-diagram-types-examples/ (дата звернення: 25.11.2024).

20. Lock A. ASP.NET Core in Action, Third Edition. Simon and Schuster, 2023. 982 p. ISBN 978-1-63343-862-0.

21. Cherny, Boris. Programming TypeScript: Making Your JavaScript Applications Scale. O'Reilly Media, 2019, pp. 45–67.

22. Angular Projects: Build Modern Web Apps Using TypeScript, Angular, and Angular Material. Packt Publishing, 2021, pp. 23–45.

23. Angular for Material Design. URL: https://link.springer.com/book/10.1007/978-1-4842-5434-9 (дата звернення: 07.05.2024).

24. Veríssimo P., Rodrigues L. Distributed Systems for System Architects. Springer Science & Business Media, 2001. 652 p. ISBN 978-0-7923-7266-0.

25. Ramirez, Sebastién. FastAPI Documentation: Build High-Performance APIs with Python. FastAPI Project, 2023, pp. 10–25.

26. Thorgersen, S., & Silva, P. I. (2021). Keycloak - Identity and access management for modern applications (pp. 15–40). Packt Publishing.

27. Kaur C., Sharma A. Social Issues Sentiment Analysis using Python. 2020 5th International Conference on Computing, Communication and Security (ICCCS)2020 5th International Conference on Computing, Communication and Security (ICCCS). (10.2020). DOI:10.1109/ICCCS49678.2020.9277251. P. 1–6.

28. Muthuswamy S. Sentiment Analysis on Twitter Data Using Machine Learning Algorithms in Python. 2018. 35p.

29. Wilken J. Angular in Action. Simon and Schuster, 2018. 415 p. ISBN 978-1-63835-600-4.

30. Чернигін Г. Л., Горбань Г. В. Система аналізу настроїв тексту на основі тематичного моделювання. Моделі, методи та засоби програмної інженерії : тези доп. Всеукр. наук.-пр. конф. «Могилянські читання – 2024» : Технічні науки. Миколаїв, 06–10 лист. 2024 р.: Чорном. нац. ун-т ім. Петра Могили, 2024. С. 94–97.

ДОДАТОК А
Апробація кваліфікаційної роботи

Результати досліджень були представлені на конференції Могилянські читання – 2024: досвід та тенденції розвитку суспільства в Україні : глобальний, національний та регіональний аспекти : XXVII Всеукраїнська науково-практична конференція : тези доповідей : технічні науки, Миколаїв, 6 – 10 листопада 2024 р. / ЧНУ ім. Петра Могили. – Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2024.

Міністерство освіти і науки України
Чорноморський національний університет імені Петра Могили
ДНУ «Інститут модернізації змісту освіти»
Південний науковий центр НАН та МОН
Інститут української археографії та джерелознавства
імені М. С. Грушевського НАН України
Первинна профспілкова організація ЧНУ ім. Петра Могили

ЧНУ

**«МОГИЛЯНСЬКІ ЧИТАННЯ – 2024:
досвід та тенденції розвитку суспільства в Україні:
глобальний, національний та регіональний аспекти»**

XXVII Всеукраїнська науково-практична конференція

ТЕЗИ ДОПОВІДЕЙ

ТЕХНІЧНІ НАУКИ

Миколаїв, 6–10 листопада 2024 року

Миколаїв – 2024

Рисунок А.1 – Обкладинка збірника тез конференції Могилянські читання – 2024

ДОДАТОК Б
Програмна реалізація сервісу ML

```python
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
import pickle

import nltk
import re
import numpy as np
from nltk.stem import SnowballStemmer
from nltk.corpus import stopwords

nltk.download('stopwords')

stemmer = SnowballStemmer('english')
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'[^a-z\s]', '', text)
    tokens = text.split()
    tokens = [stemmer.stem(token) for token in tokens if token not in stop_words]
    return ' '.join(tokens)

with open('sentiment_model.pkl', "rb") as f:
    sentiment_model = pickle.load(f)

with open('topic_model.pkl', "rb") as f:
    topic_model = pickle.load(f)

app = FastAPI()

class TextData(BaseModel):
    text: str

@app.post("/sentiment")
def predict_sentiment(data: TextData):
    try:
        processed_text = preprocess_text(data.text)
```

```
        prediction = sentiment_model.predict_proba([processed_text])[0]
        return { "sentiment": list(prediction) }
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

@app.post("/topics")
def get_topics(data: TextData):
    try:
        processed_text = preprocess_text(data.text)

        topic_model.fit([processed_text])

        feature_names = topic_model.named_steps['vectorizer'].get_feature_names_out()
        model = topic_model.named_steps['lda']

        topics = {}
        for topic_idx, topic in enumerate(model.components_):
            top_features_ind = topic.argsort()[:-3 - 1:-1]
            top_features = [feature_names[i] for i in top_features_ind]
            topics[topic_idx] = top_features

        return topics
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```